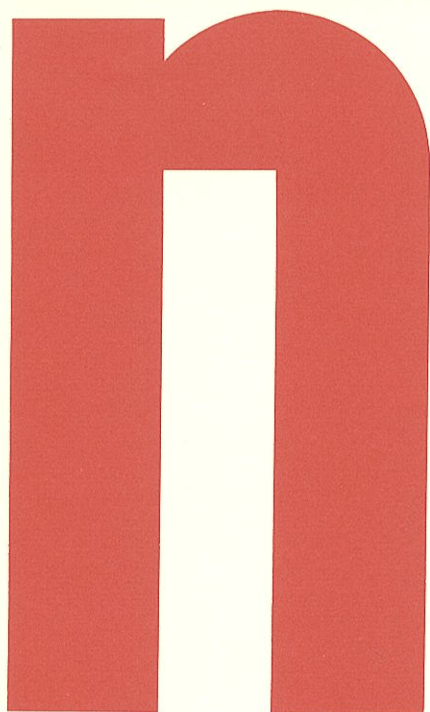
Programmation avancée
et outils pour
l'Intelligence artificielle

# NOUVEAUX ENJEUX POUR LA RECHERCHE & SES APPLICATIONS


GRECO PROGRAMMATION DU CNRS



Programmation avancée
et outils pour
l'Intelligence artificielle

# NOUVEAUX ENJEUX POUR LA RECHERCHE & SES APPLICATIONS



**GRECO PROGRAMMATION DU CNRS**
351, cours de la Libération
33045 Talence Cedex
✆ 56 84 60 89 & 56 84 60 90
Accès Minitel : 3616 GRECOTEL

AVERTISSEMENT

## De nouveaux enjeux pour la recherche & ses applications

Ce document de synthèse publié par le PRC GRECO Programmation du CNRS est une nouvelle édition du rapport scientifique "Bilan et perspectives" de Juin 1986, avec une actualisation des travaux et résultats.
Il sera largement diffusé auprès des responsables de l'ensemble de la communauté scientifique, industrielle et universitaire.
Le PRC GRECO Programmation marque ainsi sa volonté de poursuivre une politique d'ouverture et de dialogue avec tous les acteurs de la recherche et du développement technologique.

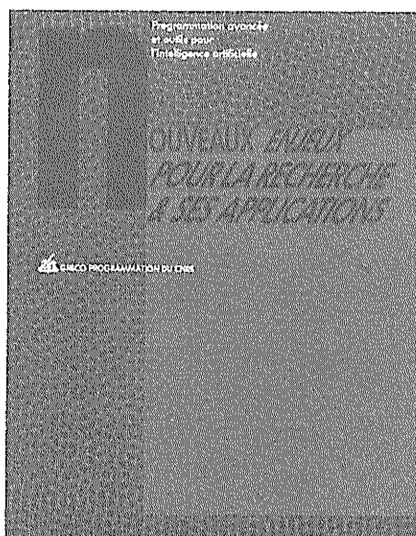
Le PRC (Programme de Recherches Coordonnées) : "*Programmation avancée et outils pour l'Intelligence Artificielle*" est financé par le Ministère de la Recherche dans le cadre du programme national mobilisateur de la filière électronique.

LA REALISATION de cette nouvelle brochure présentant l'activité des équipes du GRECO Programmation fournit l'occasion de mesurer le chemin accompli par les chercheurs en deux ans. Ainsi d'importantes avancées ont été réalisées dans le domaine de la compréhension des liens entre logique mathématique et programmation laissant entrevoir une nouvelle génération de langages de programmation, pour lesquels l'écriture d'un programme et la preuve mathématique de sa correction seront intimement liées. Quel progrès pour la fiabilité des logiciels ainsi conçus! Ce progrès ne fera que conforter les résultats déjà obtenus grâce aux spécifications algébriques, point fort des recherches du groupement.



**Robert Cori**
*Directeur du GRECO Programmation*

Dans le domaine de l'écriture de compilateurs pour les langages applicatifs et logiques, les résultats obtenus constituent une progression continue du savoir-faire. On peut maintenant réaliser des traducteurs efficaces, parfois commercialisés, permettant d'ouvrir des possibilités importantes pour tous les logiciels manipulant des données symboliques et des formalismes.
La large diffusion des stations de travail graphiques a aussi grandement favorisé l'éclosion de nouvelles formes de programmation; ainsi la programmation orientée par les objets très populaire parmi les équipes de développement de logiciel, pose des questions à la fois fondamentales et d'implémentation qui motivent les chercheurs du GRECO Programmation.




L'algorithmique enfin, considérée comme une vieille discipline dans notre domaine en pleine effervescence, a trouvé de nouveaux champs d'investigation avec les problèmes posés par la représentation et la manipulation de figures géométriques. Un nouveau chapitre de l'informatique fondamentale est en train de s'écrire.
Cette avancée remarquable de la connaissance scientifique en programmation doit être étendue aux entreprises: industriels, SSII, grands utilisateurs. Le GRECO Programmation s'est engagé dans un programme d'actions en direction des milieux professionnels en établissant de nouveaux liens entre le monde de la recherche et celui des entreprises. Une plus large diffusion des résultats a été réalisée grâce à la mise en place de structures de coopération; de nombreuses réalisations de logiciels commercialisables ont été effectuées souvent dans des PME créées par des chercheurs issus du GRECO Programmation.
Voici donc proposés dans cette brochure un certain nombre d'enjeux pour les années futures, enjeux à la fois dans le domaine des connaissances fondamentales, mais aussi dans celui de la mise à disposition pour les concepteurs et utilisateurs de logiciels d'outils sûrs et efficaces.

## REMERCIEMENTS

L'équipe de direction du GRECO Programmation
remercie tous ceux qui ont collaboré
à la réalisation de cet ouvrage.

## RÉALISATION & DIFFUSION

Ce document a été réalisé
sous la direction de Michel Mouyssinat
L'agence Ambodexter à Bordeaux
en a réalisé le design et le suivi de fabrication
AAG à Bordeaux en a assuré
la photocomposition dans différents dessins
de Futura et d'English Times
ainsi que la photogravure
Thierry Billé à Floirac a exécuté la maquette.

**Achevé d'imprimer** le 8 septembre 1988
sur les presses de l'imprimerie Sylvain à Bordeaux.
Deuxième semestre 1988.

# SOMMAIRE

1E PARTIE

# LE GRECO PROGRAMMATION DU CNRS

# Présentation du GRECO Programmation

Michel Mouyssinat
Chargé de mission
aux actions industrielles
GRECO Programmation
Bordeaux

## Une trentaine d'équipes, 250 chercheurs, des projets ambitieux.

Dans le cadre de la mise en place du programme national mobilisateur de la filière électronique et de son soutien aux équipes françaises de recherche en informatique, le Ministère de la Recherche et de la Technologie a décidé de créer au cours des années 1984 et 1985, 7 programmes de recherche (PRC : Programmes de Recherches Coordonnées), mobilisant des équipes du CNRS, des Universités, d'organismes publics ou privés autour de thèmes bien identifiés : (Intelligence Artificielle - Bases de données - Communication hommes-machines - Programmation Avancée et Outils pour l'Intelligence Artificielle - Mathématique et Informatique - Informatique et Linguistique - Coopération, Concurrence, Communication et un 8ᵉ en 1987 : Architecture. Ces programmes aux objectifs ambitieux marquaient une nouvelle étape importante pour la recherche française en informatique.

Le PRC **"Programmation Avancée et Outils pour l'Intelligence Artificielle"** a été créé en 1984 et organisé autour du GRECO Programmation (GRECO : Groupement de Recherches Coordonnées du CNRS), formation du CNRS qui regroupait déjà un grand nombre d'équipes françaises parmi les plus avancées en informatique fondamentale et jouait depuis quelques années un rôle pilote au plan national et international.

Le GRECO Programmation maintenant élargi au PRC réunit une trentaine d'équipes de recherche (CNET, CGE, IBM, INRIA, Ecole des Mines, Ecoles Normale et Polytechnique, Universités, CNRS) et vise plusieurs objectifs :

- enrichir les connaissances de ses équipes en poursuivant l'effort en faveur de l'équipement informatique, en améliorant les moyens de communication entre les chercheurs et en menant une politique d'ouverture et de coopération vers d'autres partenaires (équipes étrangères, industriels, autres PRC),
- faire sortir la programmation du stade artisanal et développer une théorie de plus en plus riche pour servir de cadre notamment au développement de programmes complexes,
- réaliser des prototypes de logiciels à des fins d'expérimentation et pour acquérir un savoir-faire dans un domaine où la France accuse encore malheureusement un certain retard,
- former par la recherche les ingénieurs et techniciens de haut niveau que réclament les entreprises et donner à l'industrie les moyens d'anticiper les conséquences des profondes mutations technologiques,
- élargir les rapports des équipes de recherche avec les entreprises industrielles pour échanger des connaissances, des savoir-faire, des expériences, valoriser les résultats des travaux de recherche et assurer leur transfert.

## Plusieurs axes de recherche sur le thème : "Programmation avancée et outils pour l'intelligence artificielle"

Le GRECO Programmation est dirigé par Robert CORI, Professeur ; son siège est installé à l'Université de Bordeaux I.

Une équipe de direction dont les membres sont choisis parmi les principaux responsables des équipes, se réunit tous les deux mois pour mettre en œuvre sa politique : définition de nouveaux projets, répartition des moyens...

Le comité de direction se réunit une fois par an pour évaluer les résultats scientifiques obtenus et proposer les axes de développement futurs.

Les recherches sont organisées en projets et regroupées en quatre pôles :

- les fondements logiques de la programmation,
- génie logiciel : conception d'environnements de programmation.
- langages et outils pour l'Intelligence artificielle et architectures de machines-langages,
- algorithmique

Le lecteur trouvera une présentation générale des pôles, réalisée par chacun des responsables, qui précède les rapports relatifs aux différents projets.

Le caractère fondamental de ces travaux place le GRECO en amont de nombreux développements de logiciels. En effet, les recherches menées par les théoriciens en logique, combinatoire, théorie des graphes, calcul formel, théorie des langages et des automates, etc. ont permis de définir de nouveaux cadres théoriques, de nouveaux modèles et corps de doctrine quiconstituent les fondements de l'ensemble des travaux actuels. Ils précèdent et sous-tendent les efforts de recherche de développement dont la finalité est de réaliser des produits. Il est en effet important de souligner qu'aujourd'hui

dans des domaines avancés de l'informatique, comme par exemple celui des bases de données, du génie logiciel, ou de l'intelligence artificielle, les produits logiciels commercialisés implémentent de nombreux concepts très théoriques et mettent en œuvre des outils algorithmiques complexes : par exemple, les langages de 4e génération d'interrogation de bases de données s'appuient sur la notion d'algèbre relationnelle dont les premières notions sont introduitent en 1962, formalisées en 1970 : un certain nombre d'outils de génie logiciel réalisent des manipulations formelles de programmes sous la forme d'arbres et utilisent donc des objets dont les représentations et propriétés algorithmiques ont été largement étudiées par les théoriciens des graphes et de la combinatoire ; le langage LISP créé en 1960, utilisé aujourd'hui dans les applications d'intelligence artificielle repose sur la théorie du lambda-calcul.

Ce sont encore les approches formelles mathématiques qui permettent, en étudiant des techniques, de découvrir les mécanismes et objets qu'elles mettent en œuvre, pour mieux pouvoir les expliquer, les comprendre et les maîtriser - parfois même elles conduiront à les automatiser. C'est ainsi, par exemple, que le développement des compilateurs est devenu aujourd'hui une tâche relativement simple et que des tentatives de production automatique de compilateurs sont en cours dans le cadre des travaux en métacompilation.

Si les exemples sont nombreux qui montrent l'importance des recherches fondamentales, ils font également prendre conscience des lenteurs avec lesquelles s'opère leur transfert vers les applications.

Ainsi, les recherches poursuivies au GRECO Programmation relèvent-elles de l'informatique fondamentale, mais celui-ci s'est aussi fixé comme règle de toujours valider l'ensemble des résultats théoriques dans le cadre de développements logiciels. Les différents projets ont donc conduit à de nombreuses réalisations de produits qui permettent ainsi en implémentant les nouveaux concepts une meilleure diffusion des acquis de la recherche et un moyen de transfert à l'ensemble de la communauté scientifique.

Pour assurer la promotion et une diffusion plus large de ces logiciels, le GRECO Programmation, bénéficiant de la dynamique de ses propres équipes, mène une politique d'ouverture et de coopération vers d'autres partenaires : équipes étrangères, autres PRC, industriels.

De nombreux contacts, concrétisés le plus souvent par le séjour de chercheurs dans des laboratoires étrangers ont donné lieu à de fructueuses collaborations et certains projets de recherche sont menés en étroite liaison avec d'autres partenaires européens ou américains (MIT, York, Berkeley, Stanford, Edimbourg, etc.)

Cette politique de coopération, par un effet de synergie, a accéléré le développement des produits logiciels, suscité de nombreuses demandes d'utilisation, favorisé leur diffusion et leurs échanges.

Leur évaluation, par les partenaires étrangers, a permis en outre de conforter la position française, aussi bien sur le plan du savoir-faire en développement logiciel que dans le domaine de la recherche fondamentale.

Le GRECO Programmation n'a pas pour objectif de capitaliser pour lui-même et ses partenaires les acquis importants de sa recherche.

Il est très désireux au contraire de s'ouvrir davantage sur tous les acteurs de la recherche et du développement technologique, d'intensifier les relations avec les milieux professionnels et de développer plus largement de nombreuses collaborations.

## Les moyens du GRECO Programmation : le serveur national GEOCUB, le réseaux des VAX, les stations de travail.

Pour atteindre ses objectifs scientifiques, le GRECO Programmation s'est doté de moyens propres. Sa politique d'équipement informatique définie en comité de direction vise à acquérir la puissance nécessaire aux besoins de calcul et à maintenir un parc homogène au niveau national, compatible avec les matériels de ses partenaires étrangers.

Un réseau de VAX système UNIX constitue son infrastructure en moyens de calcul et de communication. Le serveur national GEOCUB installé à la direction du GRECO à l'Université de Bordeaux est un des nœuds de ce réseau. Il est un point d'accès aux réseaux européens et américains et bientôt au réseau EARN.

Longtemps machine de développement pour l'ensemble des équipes, aujourd'hui équipées de stations de travail, il reste le support de la quasi-totalité des produits logiciels réalisés dans le cadre des projets ou importés des laboratoires étrangers. Il offre, de plus, des services de messagerie et de courrier electroniques qui facilitent les efforts de coopération des différentes équipes ; celles-ci ont accès en outre à tous les moyens de calcul mis en place au sein des Universités et du CNRS ; certaines d'entre elles pour répondre à des besoins plus spécifiques, sont équipées de stations de travail (SM90, SPS7, SUN).

La montée en puissance des stations de travail (SUN par exemple) et l'autonomie de plus en plus grande des équipes maintenant dotées de moyens propres, les rendent beaucoup moins dépendantes d'une machine centrale. Le GRECO Programmation reste cependant très fortement attaché à la fonction de communication en interne et avec l'extérieur. Il étudie un projet d'interconnexion des sites équipés de matériels SUN et VAX dans un réseau national dont la mise en place devrait intervenir vers la fin l'année 1988.

Le GRECO Programmation entend rester un pionnier en matière de réseau et de communication inter-sites et attache beaucoup d'importance à ce projet.

L'ensemble des moyens financiers du GRECO Programmation est constitué par les dotations du CNRS et du Ministère de la Recherche et de l'Enseignement Supérieur dans le cadre de la filière électronique (6 MF environ par an) qui viennent s'ajouter aux ressources propres de chacun des laboratoires de rattachement des différentes équipes.

## Les résultats obtenus

Le GRECO Programmation a huit ans et les résultats obtenus sont importants.

### ACCROISSEMENT DES CONNAISSANCES EN INFORMATIQUE FONDAMENTALE

L'importance des publications scientifiques dans les revues spécialisées internationales des équipes du GRECO Programmation, leurs participations nombreuses à des colloques, aux projets de recherche nationaux et internationaux, le nombre d'invitations dans les universités étrangères de ses chercheurs, la diffusion très large des logiciels développés, révèlent l'importance et la qualité des travaux effectués, la place de ses équipes dans la communauté scientifique internationale.

### FORMATION PAR LA RECHERCHE DES INGÉNIEURS ET TECHNICIENS DE HAUT NIVEAU

Le GRECO Programmation est conscient de l'importance d'une coopération étroite avec les Universités et les grandes écoles pour la formation par et à la recherche et beaucoup de ses chercheurs et ingénieurs participent à

des enseignements. Il faut souligner, à cet égard, que les résultats unificateurs et simplificateurs mis au jour par la recherche constituent des modèles qui se prêtent beaucoup mieux au transfert de connaissances qu'une somme disparate de plusieurs approches différentes des techniques informatiques.
Ces formations initiales trouvent leurs prolongements nécessaires par des actions de formation continue dans lesquelles les équipes du GRECO sont très fortement engagées.

### SAVOIR-FAIRE EN DÉVELOPPEMENT DE LOGICIELS COMPLEXES

L'ensemble des projets de recherche des différentes équipes du GRECO Programmation a conduit à de nombreuses réalisations de produits dont certains sont déjà commercialisés, d'autres à l'état de prototype. Ils sont tous accessibles sur le serveur GEOCUB installé à Bordeaux et le GRECO Programmation leur assure une large diffusion dans le cadre de la politique de coopération qu'il s'est définie. Il organise en outre de nombreuses présentations de ces logiciels accompagnées de démonstrations. Celles-ci s'adressent surtout aux professionnels pour lesquels elles offrent un réel intérêt, car les concepts et schémas théoriques issus de la recherche sont ainsi rendus très facilement accessibles. De plus, leur implémentation démontre la faisabilité de ces nouveaux systèmes et suggère de nombreux champs d'applications.



### CONNAISSANCE D'UNIX ET DE SON ENVIRONNEMENT

Il est important de rappeler que la quasi-totalité des développements logiciels est réalisée sur matériel système UNIX. Alors qu'UNIX est devenu aujourd'hui un nouvel enjeu dans la stratégie industrielle de nombreuses sociétés française, les équipes du GRECO qui en ont acquis une parfaite maîtrise contribuent très largement à mieux faire connaître ce système et son environnement dans le cadre des activités de recherche et des actions de formation (stages de formation continue, enseignement à l'Université ou dans les grandes écoles).

### TRANSFERT DES RÉSULTATS DE RECHERCHES

Les nouveaux langages, outils et méthodologies, les nouvelles architectures de machines, réalisés dans le cadre des activités de recherche, définissent les futurs standards et préfigurent les systèmes informatiques de la fin de ce siècle. L'importance, au niveau international, des grands courants scientifiques qui traversent la recherche et entraînent ces travaux, laisse en effet présager de nombreuses retombées dans le monde industriel.
Le GRECO Programmation mène un certain nombre d'actions (conférences, stages...) qui visent à diffuser la connaissance auprès de l'ensemble des milieux professionnels *(voir chapitre Actions industrielles du GRECO Programmation).*

## Quelques références de produits au stade industriel ou candidats à l'industrialisation

- Le système MENTOR de génie logiciel en cours de commercialisation par la SEMA a bénéficié des travaux de l'équipe INRIA du GRECO Programmation.
- Le langage BLAISE du terminal français STEP commercialisé par AMAIA est un produit du LRI d'Orsay.
- Un interpréteur de LOGO en LISP a été développé par le LIPT (Paris 6) dans le cadre d'un contrat avec le CNDP.
- Langage SCHEME d'intelligence artificielle développé à Paris 8 dans le cadre d'une collaboration avec la société N.C.R.
- L'interprète PROLOG/CNET implanté sur les matériels DPS et Multics des centres de calcul des Universités et de l'INRIA est actuellement commercialisé par la société CRILL sous le nom de PROLOG/P.
- Le système OASIS de manipulation de types abstraits développé par l'équipe du CNET.
- Les traducteurs PL1/C et PASCAL/ADA de l'équipe GRECO Programmation INRIA Rocquencourt sont au stade industriel.
- L'éditeur multifenêtre WINNIE du LRI d'Orsay.
- PROLOG II du GIA de Marseille pour matériel DEC, HP, SM90, IBM PC et compatibles, Apple II et Macintosh, commercialisé par la société PROLOGIA.
- Le langage Acteurs PLASMA et son extension ALOG développés au LSI de Toulouse.
- FORMES est un environnement de programmation pour la synthèse et composition musicale diffusé par la société ACT informatique (LITP Paris 6 et IRCAM).
- La société AMAIA installée à Bayonne commercialise la machine LISP MAIA, réalisée par la CGE dans le cadre des travaux menés par les équipes du CNRS, de l'INRIA et du CNET.
- Le-LISP de l'INRIA commercialisé par la société ILOG.
- LORE est un logiciel prototype de programmation par objets développé par les laboratoires de la CGE à Marcoussis et diffusé par la société ISR.
- Le langage CAML diffusé par la société ILOG.
- Le langage LPG de programmation générique.
- ASPEGIQUE pour la manipulation de types abstraits.
- Le langage REVE de déduction automatique.
- Méta-Vlisp et Méta-Objets, opérationnels notamment sur PC et SUN3, ont été commandés par le CNDP (Centre National de Documentation Pédagogique).
- ADA est au centre de nombreux projets à l'INRIA - (SOPHIA-ANTIPOLIS). Les logiciels suivants sont au stade pré-industriel ou en cours de réalisation :
GRAPHADA éditeur syntaxique graphique pour ADA.
ADA+ un préprocesseur orienté objets pour ADA.
ADAGE un analyseur syntaxique compatible YACC, attribué, en ADA, générant du code ADA.
PROLOG-ADA un interprète PROLOG à stratégies paramétrables écrit en ADA, interfaçable avec des packages ADA.
- Une version portable du standard LISP industriel COMMON-LISP sous UNIX écrite en C a été portée sur VAX, SUN, et SM90 (Patrick Greussay - Université Paris VIII).



**Réunion de l'équipe de direction élargie**

*Pierre Lescanne — Michel Mouyssinat — Paul Franchi-Zannettacci — Patrick Henry — Patrick Sallé — Emmanuel Saint-James — Michel Bidoit — Robert Cori*





## L'équipe de direction

**M. Bidoit** : *L.R.I., Orsay;* **C. Choffrut** : *LITP, Paris 7;* **A. Colmerauer** : *GIA Marseille.*

**R. Cori** : *Univ. Bordeaux I, Directeur du GRECO Programmation;* **G. Cousineau** : *ENS, rue d'Ulm.*

**P. Flajolet** : *INRIA, Rocquencourt;* **P. Franchi-Zannettacci** : *INRIA-ISI, Sophia-Antipolis.*

**P. Greussay** : *Univ. Paris 8;* **P. Lescanne** : *CRIN, Nancy;* **P. Salle** : *ENSEEIHT, Toulouse*

Les personnes suivantes sont à la disposition des chercheurs et des industriels au siège du GRÉCO Programmation à Bordeaux :

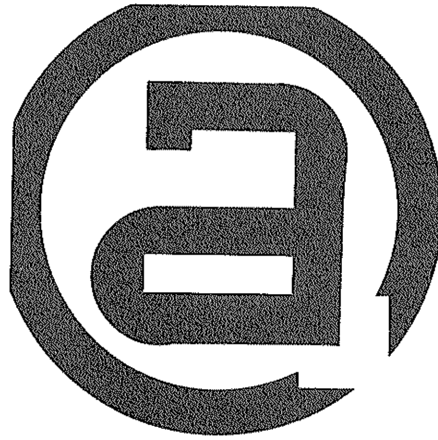
**Patrick Henri,** Ingénieur au CNRS pour tout ce qui concerne les aspects matériels, systèmes et réseaux;

**Monique Claverie,** pour l'organisation administrative;

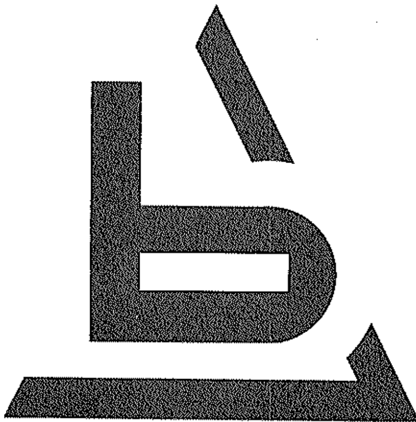
**Marie-France Dudon,** pour les Actions Industrielles : secrétariat.

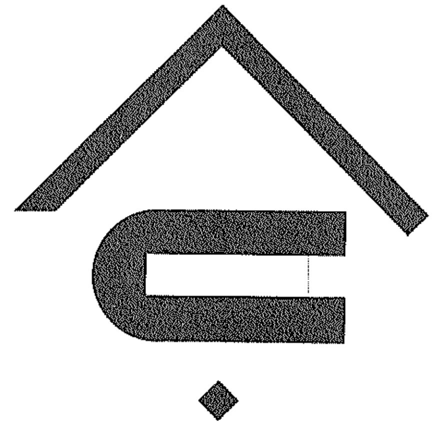


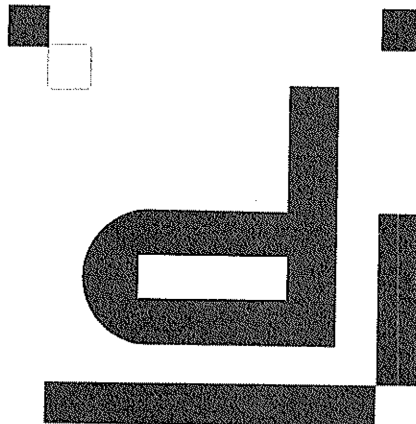
*Maurice Nivat au cours d'une réunion de l'équipe de direction élargie.*











# FONDEMENTS LOGIQUES DE LA PROGRAMMATION



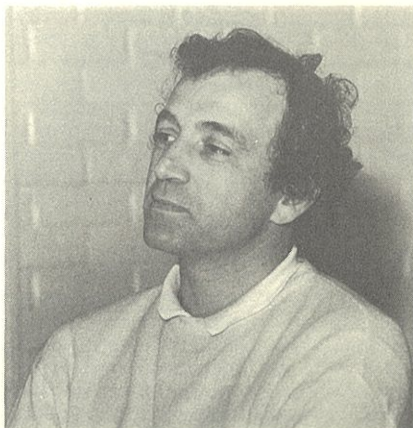
ROBERT MORRIS
*"LABYRINTH", 1974, CONTRE-PLAQUÉ, MASONITE, PEINTURE A L'HUILE,
DIAMÈTRE : 914,5 cm, HAUTEUR : 244 cm.
COLLECTION PANZA DI BIUMO, MILAN.*

Pierre Lescanne




## Logique et sémantique de la programmation

Dès son origine, la programmation est clairement apparue comme une fille de la logique mathématique et les grands progrès des logiciens des années 30 comme Church, Curry, Gödel, Tarski et Turing ont immédiatement précédé la naissance des premiers ordinateurs et cela n'est pas fortuit. Si une longue période de déconnexion entre la logique et l'informatique a suivi, c'est parce que les impératifs de mise en œuvre occultaient la nature mathématique sous-jacente des processus. Ce fut la période de la bataille du "goto". Certes, des pionniers, comme Landin, McCarthy, Strachey ou Scott proposaient tôt de fonder les langages sur la logique et, d'autre part, la déduction automatique fut rapidement une utilisatrice de logique et d'informatique, mais on n'en était pas encore au consensus actuel où l'on n'imaginerait pas de définir un langage de programmation sans étudier les concepts logiques profonds. La sûreté est devenue un impératif et seuls des langages puissants et logiquement sains permettent d'atteindre cet objectif. Tout cela a conduit à une interaction de plus en plus grande entre la communauté des logiciens et celle des informaticiens. Le programme de recherche concertée Programmation Avancée et Outils pour l'Intelligence Artificielle est un acteur essentiel de ce renouveau. On trouve, en fait, représentées toutes les grandes tendances actuelles, à l'exception de certains aspects ayant trait spécifiquement au parallélisme qui ont trouvé leur place au sein d'un autre programme : $C^3$.



- Les logiques d'ordres supérieures, notamment la théorie des constructions, constituent, en effet, la base des travaux du groupe FORMEL, qui étudie aussi ses implications dans la définition et l'implémentation du langage CAML.
- La logique modale a donné lieu à un nouveau langage de programmation d'un autre genre, MOLOG, ainsi qu'au développement d'outils de démonstration automatique spécifiques.
- La logique des clauses de Horn, qui sous-tend PROLOG, est étudiée par le groupe METHEOL. Celui-ci propose un certain nombre d'outils à la fois théoriques et pratiques qui permettent une meilleure compréhension et une meilleure définition du langage ainsi qu'une plus grande efficacité des ses mises en œuvre.
- La logique équationnelle joue un rôle important dans les langages de spécification modernes et a suscité des implantations de langages comme OBJ et des environnements de déduction automatique comme REVE.
- C'est encore de logique qu'il s'agit quand l'équipe ATTRISEM propose des algorithmes efficaces et généraux pour les attributs sémantiques.

Toutes les équipes du pôle logique et sémantique ont en commun le souci d'allier la réflexion théorique à l'implantation de logiciels correspondants. Les présentations qui vont suivre montrent comment cette harmonie se réalise.

# Outils et Méthodes Théoriques pour la Programmation en Logique

METHEOL *MOTS CLES*
démonstration de programmes
intelligence artificielle
langages de cinquième génération
logique trivaluée
programmation par règles
prolog
sémantique logique
sémantique opérationnelle

## La programmation par règles

Depuis quelques années, un nouveau type de programmation est apparu, fondé sur la notion de règle. Les règles en question peuvent être simplement de la forme :

si CONDITION alors CONCLUSION

ce qu'on écrit parfois :

CONCLUSION si CONDITION

Il s'agit là d'un formalisme très général utilisé dans les langages de représentation de connaissances pour systèmes experts, dans la programmation logique et ses extensions (Prolog avec négation par exemple), et dans les bases de données déductives.

Les problèmes posés par la programmation avec des règles sont très nombreux et liés à des considérations abstraites qu'il est indispensable de maîtriser faute de quoi les systèmes de programmation par règles sont inconsistants ou inefficaces (ou les deux à la fois !).

Le but de l'équipe METHEOL est de travailler à l'éclaircissement et au développement des concepts sur lesquels peut s'appuyer la programmation logique par règle de manière à en tirer les conclusions pratiques (modèles d'implémentation, méthodologies). Ces concepts sont à la fois théoriques (études des liens formels entre de tels modèles de programmation et la logique mathématique) et d'application concrète (efficacité des interpréteurs de règles, compilation des programmes à base de règles, méthodologie de la programmation avec des règles).

## Sémantique de la programmation par règles

L'intérêt de la programmation par règles réside dans la possibilité d'interpréter les unités de base d'un programme (les règles) comme ayant un sens intuitif logique. Les fondements de la programmation par règles doivent donc se trouver dans la logique et plus particulièrement dans la logique mathématique telle qu'elle a été développée depuis le début du siècle. Les concepts de base seront donc ceux de modèles et de systèmes formels de démonstration.

Les nombreux travaux faits entre 1965 et 1975, en démonstration automatique autour de la résolution de Robinson, fournissent un point de départ qui, exploité dans les travaux de Hill, Kowalski, Van Emden et Apt (entre 1974 et 1982), sont maintenant universellement reconnus comme fournissant les bases de toute réflexion théorique sur la programmation par règles.

Mais ces travaux laissent en suspens d'assez nombreux points, notamment ceux qui concernent la négation. Les travaux des chercheurs du groupe METHEOL ont sensiblement fait avancer la vision sémantique qu'on peut avoir de la programmation logique avec négation. A titre d'exemple, nous citerons les deux contributions suivantes :

- le modèle de Ferrand et Deransart ([DeF 87]) qui s'appuie sur la notion d'univers de Herbrand avec variables et d'arbre de preuves. Il permet d'avoir une vision unifiée des différents niveaux de sémantique qu'il est possible d'associer à un programme Prolog avec négation (sémantique constructive, sémantique logique, sémantique opérationnelle non déterministe, sémantique opérationnelle déterministe). Ce modèle sert aujourd'hui de base à la définition d'une spécification formelle de Prolog pour les comités de normalisation de Prolog : BSI et AFNOR (voir [DeR 87]).
- le modèle de la logique trivaluée ([Del 87a] [Del 87b]) fondé sur un nouveau type de sémantique logique (dit "à la Kripke"). Il donne une vue générale très simple de la programmation avec règles et négation permettant de retrouver pratiquement toutes les bonnes propriétés de la programmation avec règles sans négation (existence de plus petit modèle, calculabilité en temps raisonnable des plus petits modèles).

## Analyse du contrôle en programmation par règles

La programmation par règles, dont l'idée de départ est d'être la plus déclarative possible, rencontre bien sûr des difficultés dans le contrôle du déroulement des programmes. Celles-ci sont traitées en Prolog par des moyens "très peu sémantiques" : fixation une fois pour toutes d'une stratégie d'exploitation des règles (dont le programmeur doit tenir compte pour écrire ses programmes) et utilisation d'opérateurs n'ayant un sens que purement opérationnel (comme le CUT de Prolog).

Il s'agit d'un des thèmes de travail des chercheurs du groupe METHEOL. Sur ce projet, les travaux de M. Billaud ([Bill 66]) ont apporté un éclaircissement essentiel. Les méthodes qu'il propose permettent de comparer la puissance des opérateurs de contrôle et de montrer que l'opérateur si-alors-sinon (sémantiquement assez propre) peut être substitué dans presque tous les cas au CUT (dont l'utilisation détruit le sens déclaratif des programmes logiques).

L'implémentation d'un Prolog fondé sur ce modèle a été réalisée.



## Références


[Bil 85] M. Billaud, *"Une formalisation des structures de contrôle en Prolog"*, Thèse de 3e cycle, Univ. de Bordeaux I, 1985.

[Bil 86] M. Billaud, *"Prolog control structures: a formalisation and its application"*, Publ. UER Math. Info, Université de Bordeaux.

[Cod 86a] Ch. Codognet, P. Codognet et G. Filé, *"A very intelligent backtracking method for logic programs"*, Proc of the ESOP, LNCS 213, 1986, pp. 315-326.

[Cod 86b] Ch. Codognet, P. Codognet et G. Filé, *"Backtracking intelligent en programmation logique"*, Actes du Séminaire de Programmation en Logique de Trégastel, 1986, pp. 25-50.

[Cod 86c] Ch. Codognet, P. Codognet et G. Filé, *"Optimization of Logic Program Execution Build on their static analysis"*, R.R. 86-24, Université de Bordeaux.

[Cou 87] B. Courcelle and P. Deransart, *"Proof of partial correctness of attribute grammars with application to recursive procedures and logic programs"*, Information and Control, 1987.

[Del 86a] J.-P. Delahaye, *"Outils Logiques pour l'Intelligence Artificielle"*, Ed. Eyrolles, Paris, 1986. (Traduction Anglaise Kogan Page 1987.)

[Del 86b] J.-P. Delahaye et P. Paradinas, *"Définitions de stratégies équitables en programmation logique"*, Actes du Séminaire de Programmation en Logique de Trégastel, 1986, pp. 7-24.

Bordeaux
Lilles

**Responsable scientifique**
Jean-Paul Delahaye

**Organismes**
Université Sciences et Techniques Lille
U.E.R.I.E.A.A. - M3
56655 Villeneuve-d'Ascq
☎ 20 43 47 22/20 44 17 29
Université de Bordeaux I
Bull
I.N.R.I.A. Rocquencourt
Université d'Orléans

**Participants**
B. Courcelle
M. Billaud
Ph. Codognet
Ch. Codognet
S. Yoccoz
M. Corsini
M. Dauchet
G. Comyn
J.-P. Delahaye
Ph. Devienne
P. Lebegue
N. Caridroit
P. Deransart
G. Ferrand
M. Bergère
L. Lardeau
G. Richard
J.-M. Kerisit
J.-M. Pugin

## Étude statique et dynamique des programmes logiques

L'analyse des programmes logiques (et surtout des programmes Prolog) est l'un des problèmes majeurs qui se posent aujourd'hui à propos de Prolog.

Elle doit permettre une meilleure maîtrise de la programmation elle-même : problème de méthodologie de la programmation, calcul des modes, démonstration de correction, de complétude et de terminaison des programmes (travaux de P. Deransard, P. Devienne, M. Dauchet, B. Courcelle).

Cette analyse constitue un moyen essentiel pour l'amélioration de la compilation et de l'exécution des programmes logiques. Les travaux réalisés par Ph. et Ch. Codognet (en collaboration avec G. Filé de l'Université de Padou) et par M. Corsini sont sur ce point très importants. L'implémentation de leurs méthodes de "backtracking intelligent" a permis de montrer qu'à l'exécution, certains programmes écrits "naïvement" (c'est-à-dire d'une manière déclarative simple, sans utilisation d'artifices de contrôle extra-logiques) sont aussi efficaces que ces mêmes programmes additionnés des directives de contrôles adéquates. Ces résultats présentent un grand intérêt car ils montrent que le paradigme d'une programmation par règles, laissant à l'interpréteur de règles ou au compilateur de règles la liberté du contrôle et permettant au "programmeur" de ne pas sortir d'une pensée logique pure, est à prendre au sérieux encore aujourd'hui (ce que certains problèmes rencontrés avec Prolog ont parfois mis en doute).

## Prospectives

L'importance de la programmation par règles ne semble pas devoir être remise en cause au cours des prochaines années et bien au contraire, elle apparaît comme en plein essor, en particulier en Intelligence Artificielle. Les travaux du groupe METHEOL qui, à partir d'une réflexion fondamentale, tentent d'affiner les modèles de la programmation logique, d'en préciser la méthodologie et d'en améliorer la compilation et l'exécution sont situés au cœur de cette problématique. Ces travaux, dont il faut souligner l'importance de l'apport théorique, conduisent en outre à de nombreux développements. Parmi ceux en cours, on peut citer : un prolog à stratégies équitables (permettant une meilleure gestion de la négation et limitant les boucles), un prolog intégrant les méthodes d'analyse de programmes de M. Dauchet, P. Devienne et P. Lebegue, un générateur de systèmes experts fondé sur la logique trivaluée et un prolog avec backtracking intelligent. Ces implémentations concrètes d'un intérêt pratique évident montrent, s'il en était encore besoin, que la coopération théorie-réalisation est le meilleur moyen d'avancer efficacement.

[Del 87a] J.-P. Delahaye, *"Sémantique logique et dénotationnelle des interpréteurs prolog"*, Theoretical Informatics, 1988.

[Del 87b] J.-P. Delahaye, *"Effets de l'utilisation du coupe-choix sur la sémantique déclarative de Prolog"*, Séminaire de Programmation Logique de Trégastel, 1987.

[Del 87c] J.-P. Delahaye, *"Sémantique et complétude du chaînage avant en calcul propositionnel"*, 7es Journées Internationales sur les Systèmes Experts et leurs Applications, Avignon, 1987.

[DeF 87] P. Deransart et G. Ferrand, *"An Operational Formal Definition of Prolog"*, Symposium on Logic Programming, San Francisco, 1987.

[DeR 87] P. Deransart et G. Richard, *"Formal Specification of Prolog"*, Séminaire de Programmation Logique de Trégastel, 1987.

[Der 85] P. Deransart and J. Maluszynski, *"Relating logic programs and attribute grammars"*, J. of Logic Programming, 1986, 2, pp. 119-155.

[Der 87] P. Deransart et G. Ferrand, *"Détection d'erreurs en programmation logique : réalisation expérimentale"*, J. of Logic Programming, 1987.

[Dev 86] Ph. Devienne et P. Lebegue, *"Weighted graphs: a tool for logic programming"*, Proc of the CAAP, Nice, LNCS 214, 1986, pp. 100-111.

[Ker 86] J.-M. Keresit, J. Rohmer and R. Lescœur, *"The Alexander Method"*, New Generation Computing, 4, 1986, pp. 273-285.

# Etude de la Réécriture et de ses Applications

EURECA *MOTS CLES*
analyse d'algorithmes
démonstration automatique
langage de spécification
méthodes de spécification
OBJ (langage orienté objet)
réécriture
réécriture conditionnelle
spécifications algébriques
spécifications formelles
spécifications par types abstraits
types abstraits algébriques
unification
validation de spécifications
rêve



## Références


[1] F. Bellegarde, *"Rewriting systems on FP expressions to reduce the number of sequences yielded"*, Science of Computer Programming, 6:11-34, 1986.

[2] F. Bellegarde and P. Lescanne, *"Transformation orderings"*, in *12 th Coll. on Trees in Algebra and Programming, TAPSOFT*, Springer Verlag, 1987.

[3] S. Benachi, *"Algorithmes sur les ensembles définis par générateurs et relations"*, rapport de DEA de l'Université de Nancy 1, 1987.

[4] A. Ben Cherifa and P. Lescanne, *"An actual implementation of a procedure that mechanically proves termination of rewriting systems based on inequalities between polynomial interpretations"*, in J. Siekmann, editor, *Proc. 8th Conference on Automated Deduction, Oxford*, pages 42-51, Springer Verlag, 1986.

[5] A. Ben Cherifa and P. Lescanne, *"Termination of rewriting systems by polynomial interpretations and its implementation"*, Science of Computer Programming, 9(2):137-160, October 1987.

[6] W. Bousdira and J.-L. Rémy, *"Complétion des systèmes de réécriture conditionnelle"*, in *Actes des Journées GROPLAN 1987*, à paraître dans la revue BIGRE+GLOBULE, Aix-en-Provence, 1987.

[7] W. Bousdira and J.-L. Rémy, *"Hierarchical contectual rewriting with several levels"*, Technical Report, Centre de Recherche en Informatique de Nancy, 1987.


Dans de nombreuses applications de l'informatique, telles la sécurité d'accès dans les réseaux, l'aéronautique ou plus généralement le contrôle de processus complexes, le problème de la correction du logiciel est primordial. Le résoudre constitue un défi scientifique et technique. Les solutions pragmatiques (tests) sont inadéquates, car si elles sont effectuées sur le produit final ou presque c'est, en général, bien tard. Tout le monde s'accorde à penser que la solution se trouve dans une discipline de programmation très rigoureuse du cahier des charges à la maintenance. Cette discipline requiert une phase de spécification où le programmateur exprime le problème qu'il a à résoudre dans un langage que lui ou ses collègues doivent pouvoir relire aisément, mais aussi que l'ordinateur peut traiter. Pour cela, il utilise un formalisme rigoureux et non ambigu. Par leur rigueur, les spécifications ressemblent aussi bien à des programmes qu'à des énoncés mathématiques. Leur but est de fournir un document formel qui exprime complètement le problème de l'utilisateur et qui sera utilisé tant dans la phase de programmation que lors de la maintenance. En plus, on exige que ces spécifications fassent l'objet de vérifications rigoureuses, similaires à des preuves mathématiques, afin d'être certain qu'elles correspondent bien à ce que l'on veut. Ces preuves doivent pouvoir être réalisées par un ordinateur ou tout au moins en interaction avec lui. Seul un langage formel permet une telle interaction. Il est bon que les spécifications soient exécutables ; ainsi, l'utilisateur obtiendra dès la phase de spécification un prototype de son système qui permettra les premières évaluations et fera très tôt apparaître les vices éventuels.

Le formalisme des types abstraits algébriques permet cette rigueur et ces vérifications. Un type abstrait spécifie une structure de données abstraite, c'est-à-dire se contente de préciser quelles sont les données et les opérations agissant sur ces données. Les propriétés caractéristiques de ces opérations sont décrites par des axiomes qui sont des formules de logique égalitaire ou du premier ordre en général. Les spécifications sont exécutables si la logique considérée possède un système de déduction complet qui permet de calculer dans cette logique. C'est le cas en particulier pour la logique égalitaire.

Dans l'optique de construire des outils pour la démonstration automatique de théorèmes qui vont servir à valider les spécifications, l'activité de recherche de l'équipe s'est focalisée sur quatre domaines principaux.

*LA RÉÉCRITURE ET LA DÉMONSTRATION AUTOMATIQUE.* La technique sur laquelle s'appuie l'équipe est la réécriture. Outre la poursuite de l'étude théorique de cet outil, le principal apport concerne l'exploitation des techniques de réécriture pour la démonstration automatique en logique du premier ordre.

*LA RÉÉCRITURE CONDITIONNELLE.* La technique de réécriture conditionnelle qui permet de prendre en compte des équations soumises à des conditions connaît d'importants développements théoriques actuellement. Le fait marquant dans l'équipe est un travail de synthèse entre plusieurs approches.

*LES LANGAGES DE SPÉCIFICATIONS BASÉS SUR LA RÉÉCRITURE.* Un langage de spécification exécutable modulaire, appelé *OBJ*, a été proposé par le groupe de J. Goguen au SRI. Des membres de l'équipe ont participé à l'étude et à l'implantation de la version 3 du logiciel. Leur recherche se centre sur l'étude du cadre logique et des outils de validation nécessaires à ce type de langage.

*ANALYSE D'ALGORITHMES.* La recherche menée par le groupe porte sur l'évaluation en moyenne d'algorithmes opérant notamment dans un contexte dynamique. L'analyse mathématique permet de trouver les modes de représentation les plus efficaces pour les structures de données.

## Bilan des actions de recherche

La recherche menée dans le groupe est présentée selon les quatre volets déjà mentionnés, à savoir : la réécriture et la déduction automatique, la réécriture conditionnelle, les langages de spécifications basés sur la réécriture, l'analyse d'algorithmes.

### 1. RÉÉCRITURE ET DÉDUCTION AUTOMATIQUE

(Responsable : Pierre Lescanne; participants : Ahlem Ben Cherifa, Françoise Bellegarde, Isabelle Gnaedig, Claude Kirchner, Hélène Kirchner, Jalel Mzali, Pierre Réty, Michaël Rusinowitch.)
Rappelons que l'équipe a développé le logiciel *REVE*, actuellement distribué dans une trentaine de laboratoires dans le monde. *REVE* apporte déjà des outils originaux dans le domaine de la terminaison des systèmes de réécriture, de la réécriture équationnelle, de la preuve par récurrence, des méthodes complètes de preuves dans le cas équationnel.

Les travaux récents sur la réécriture et la déduction automatique ont porté dans quatre directions :

- les preuves de terminaison des systèmes de réécriture,
- l'unification et la disunification,

Nancy

**_Responsable scientifique_**
Pierre Lescanne

**_Organisme_**
CRIN/Université de Nancy
BP 239
54506 Vandœuvre-lès-Nancy Cedex
☎ 83 91 21 19

**_Participants_**
Mikulàš Hermann
Claude Kirchner
Hélène Kirchner
Jean-Luc Remy
Isabelle Gnaedig
René Schott
Ahlem Ben Cherifa
Françoise Bellegarde
Aristide Mègrelis
Jalel Mzali
Bruno Randrianarimanana
Michaël Rusinowitch
Jieh Hsiang
Saad Benachi
Wadoud Bousdira
Azzedine Lazrek
Pierre Réty
Jocelyne Rouyer
Stefan Uhrig

- la confluence,
- les méthodes de démonstration automatique fondées sur les méthodes de la réécriture.

D'autre part, un effort en direction de la diffusion des connaissances a été fait par la diffusion des deux articles de synthèse sur le sujet [19,29].

### *1.1. LA TERMINAISON DES SYSTÈMES DE RÉÉCRITURE*

La terminaison des systèmes de réécriture est un problème essentiel. En effet, ces études conduisent à la preuve de terminaison des programmes fonctionnels, problème important par lui-même ; mais elles interviennent aussi de façon décisive dans la procédure de complétion de Knuth et Bendix. Le problème est indécidable, par conséquent, il n'y a pas d'algorithme qui permette de prouver la terminaison des systèmes de réécriture. En revanche, des méthodes particulières ont été définies et *REVE* en implante un certain nombre.

Pour les non-spécialistes, la méthode de preuve de terminaison fondée sur des *valuations polynômiales* est la plus populaire, bien que n'étant pas la plus simple à utiliser. Elle n'avait cependant jamais reçu d'implantation convaincante. Ahlem Ben Cherifa et Pierre Lescanne ont proposé une méthode de preuve fondée sur des principes simples qui vient à bout de pratiquement tous les systèmes pour lesquels nous connaissions une preuve par cette méthode, preuve en général bien fastidieuse à la main [4,5]. Cette méthode a été implantée dans *REVE*. En fait, une version simplifiée du problème peut se ramener à montrer qu'un polynôme à plusieurs variables ne prend que des valeurs positives sur $\{(x_1,...,x_p) \quad \mathbf{R}^p | x_i > 1\}$. Des études fondées sur la méthode de Sturm sont menées pour obtenir si possible un algorithme de décision [39]. Une approche fondée sur la transformation du système de réécriture par un autre système de réécriture a aussi été proposée par Pierre Lescanne et Françoise Bellegarde [2]. Les deux systèmes de réécriture doivent satisfaire entre eux une propriété de *cohérence* qui s'apparente à la confluence et se teste de manière similaire. Un problème important en réécriture, ces dernières années, a été l'extension des méthodes de preuve de terminaison classiques au cas de la réécriture équationnelle. Isabelle Gnaedig, en collaboration avec Pierre Lescanne, a développé un ordre permettant de prouver la terminaison de la réécriture modulo un ensemble d'axiomes associatifs-commutatifs. Le principe de base, dû à D. Plaisted, est la transformation, par un système de réécriture annexe, du système dont on veut prouver la terminaison. Une preuve complète de la validité de cette méthode a été établie dans [16]. La méthode des valuations polynômiales proposée ci-dessus [5] s'applique aussi très bien au cas associatif et commutatif et c'est actuellement la seule méthode proposée dans *REVE*.

### *1.2. L'UNIFICATION ET LA DISUNIFICATION*

L'unification est au centre de la déduction automatique. Les travaux de Claude Kirchner ont conduit à une méthode de résolution d'équations modulo des axiomes qui fonctionne dans de nombreux cas [23]. Il a mis par ailleurs en évidence une classe de théories dites syntaxiques pour lesquelles un algorithme de décision de l'unification se déduit automatiquement de la syntaxe des axiomes [21]. Une spécification complète de la méthode en *OBJ* a été écrite (voir le paragraphe sur les réalisations en *OBJ*). La méthode générale a été également appliquée à la combinaison d'algorithmes d'unification, et a en particulier conduit à la conception d'un nouvel algorithme d'unification associative-commutative [22]. Une autre méthode d'unification équationnelle a été étudiée, appelée *surréduction*. Elle est essentiellement fondée sur une variante de la réécriture qui utilise l'unification classique là où on utilisait le filtrage [37,33,36,38]. On réécrit l'équation à résoudre en faisant apparaître un sous-terme à réécrire grâce à une unification. En cumulant les substitutions introduites de proche en proche à chaque étape, on construit la solution finale.

Le problème voisin de la *disunification* a été abordé. Il s'agit de la résolution de systèmes où apparaissent, outre les égalités habituelles de l'unification, des diségalités, c'est-à-dire des expressions de la forme $s \neq t$. Une solution fondée sur un ensemble de règles d'inférence a été proposé [9,26]. Des algorithmes de filtrage ont été aussi étudiés.

Par ailleurs, Claude Kirchner a organisé une rencontre internationale sur l'unification qui a eu lieu au Val-d'Ajol dans les Vosges [24].

### *1.3. LA CONFLUENCE*

La procédure de complétion de Knuth et Bendix est utilisée pour produire un système de réécriture confluent et noethérien permettant de décider de l'égalité dans une théorie équationnelle, c'est-à-dire pour produire un algorithme capable de prouver ou de réfuter un théorème équationnel.

Malheureusement, il existe de nombreux cas où cette procédure diverge en produisant une infinité de règles que l'on peut cependant décrire finiment. Des études de l'intéterminisme de la complétion vis-à-vis de l'ordre choisi, tantôt divergeant, tantôt produisant un système en un temps fini, ont été réalisées grâce à *REVE* et développées sur des exemples [14, 30]. Hélène Kirchner a étudié comment malgré tout définir un concept de réécriture à l'aide d'une infinité de règles. Elle utilise pour cela des *méta-règles*. Par ce moyen, elle obtient une méthode de décision pour la théorie équationnelle sous-jacente [27].

La complétion est aussi utilisée pour les preuves par récurrence dans les théories équationnelles [28].

Françoise Bellegarde a étudié comment des systèmes confluents et noethériens pouvaient être appliqués à la transformation de programmes fonctionnels [1].

Dans un registre un peu différent, il a été prouvé que le problème de la confluence des systèmes de réécriture de termes clos, c'est-à-dire sans variables, est décidable [10].

### *1.4. LES MÉTHODES DE DÉMONSTRATION AUTOMATIQUE FONDÉES SUR LA RÉÉCRITURE.*

Par une nouvelle méthode fondée sur des *arbres sémantiques transfinis*, Michaël Rusinowitch [40] a pu prouver la complétude de stratégies de démonstration automatique connues depuis longtemps, mais dont la preuve de complétude résistait, faute d'outils théoriques adéquats. Ces stratégies sont liées à la *paramodulation* qui est une règle d'inférence permettant de prendre en compte l'égalité, Michaël Rusinowitch montre comment les techniques de réécriture peuvent améliorer ces stratégies. Il a notamment, en collaboration avec Jieh Hsiang, professeur à l'Université de Stony Brook (State University of New York), mis au point une *procédure de complétion sans échec* [7] qui a été implantée par Jalel Mzali [32]. Cette procédure est sans échec, car si elle ne peut pas orienter une équation en règle de réécriture, cause habituelle de l'échec, elle continue son exécution en utilisant les équations de façon non orientée. La preuve de complétude montre que cette stratégie fournit une procédure de semi-décision pour toute les théories équationnelles, c'est-à-dire une procédure qui prouve mais ne réfute pas forcément. Des règles d'inférence pour les axiomes de régularité ont aussi été décrites [18].

## 2. RÉÉCRITURE CONDITIONNELLE

(Responsable : Jean-Luc Rémy ; participants : Wadoud Bousdira, Hélène Kirchner, Stefan Uhrig).

Les recherches sur les systèmes de réécriture conditionnelle se sont poursuivies au sein du projet, avec pour support le logiciel expérimental *REVEUR-4*. L'effort a porté principalement sur trois axes de travail :

1. Amélioration du logiciel *REVEUR-4*, pour augmenter en particulier le nombre de fonctions qu'il est capable de prendre en compte.

2. Extension de l'étude théorique des systèmes de réécriture conditionnelle à plusieurs niveaux de hiérarchie.

3. Réalisation d'un travail de synthèse entre plusieurs approches, en particulier celle des systèmes hiérarchiques et celle des systèmes réduisants.

### *2.1. RÉALISATIONS LOGICIELLES*

Le logiciel *REVEUR-4* permettait jusqu'à maintenant de tester la confluence sur les termes clos de systèmes hiérarchiques à deux niveaux, et de vérifier, en outre, une propriété dite de bonne couverture des préconditions des règles. L'effort de cette année a porté sur le problème de la complétion d'un système qui n'est pas confluent sur les termes clos. Une réalisation a été effectuée; elle a permis de compléter plusieurs systèmes hiérarchiques associés à des spécifications algébriques de types abstraits. De l'autre côté, comme il s'agit d'un problème difficile, la réalisation effectuée a également permis de mieux comprendre dans quelles directions faire porter nos efforts. La nouvelle version du logiciel *REVEUR-4* a été présentée au cours d'un exposé aux journées AFCET-GROPLAN à Aix-en-Provence (Février 1987) [6] et dans des sessions de démonstration à la conférence STACS à Passau (RFA) (Février 1987) [8].

### *2.2. EXTENSION DE L'ÉTUDE THÉORIQUE A PLUSIEURS NIVEAUX DE HIÉRARCHIE*

Il est naturel de vouloir considérer un nombre arbitraire de niveaux de hiérarchie. Cela nécessite une étude très minutieuse des mécanismes en présence. En particulier, plusieurs propriétés entrent en ligne de compte et il est nécessaire de les supposer acquises au niveau $n-1$ pour être en mesure de vérifier qu'elles persistent au niveau $n$. Également, il est apparu en cours d'étude qu'il pouvait être nécessaire de considérer, pour une propriété donnée, de nombreuses variantes de celle-ci, selon que l'on se place, par exemple, dans le cadre équationnel ou dans le cadre opérationnel (c'est-à-dire avec des règles de réécriture) et dans ce dernier cas, selon que l'on utilise les règles sans restrictions ou bien avec des restrictions hiérarchiques. Une comparaison approfondie de ces variantes a été menée. Le résultat final de cette étude est un théorème donnant des conditions suffisantes de confluence sur les termes clos pour un nombre arbitraire de niveaux de hiérarchie. Il fait l'objet d'un article à paraître [7].

### *2.3 TRAVAIL DE SYNTHÈSE ENTRE LES APPROCHES DES SYSTÈMES HIÉRARCHIQUES ET DES SYSTÈMES RÉDUISANTS*

Alors que l'approche des systèmes hiérarchiques, commencée à Nancy, a constitué le premier effort conséquent pour entreprendre l'étude des systèmes de réécriture conditionnelle, une deuxième approche, celle des systèmes réduisants, initiée par Stéphane Kaplan à Orsay, s'est avérée également intéressante. Jusqu'au début de 1987, ces deux approches avaient évolué de manière indépendante, bien que leurs promoteurs respectifs se soient tenus mutuellement au courant de leurs travaux. Cette année, un travail commun de synthèse a été entrepris par ces derniers. Il a par exemple montré que le concept de forme normale contextuelle, développé d'abord dans le cadre des systèmes hiérarchiques, s'adapte au cadre des systèmes réduisants. Ce concept est l'extension naturelle du concept de forme normale au cas conditionnel. Des lumières importantes ont également été apportées en ce qui concerne la nature des spécifications conditionnelles dans lesquelles les préconditions sont des expressions de sorte booléenne. Ce problème avait pour l'essentiel été laissé dans l'ombre jusqu'à présent dans la mesure où les systèmes donnés en exemple se comportaient toujours comme on pouvait s'y attendre. Sans entrer dans les détails, disons que deux propriétés doivent être satisfaites pour que l'on soit assuré d'un comportement correct : *la complétude suffisante et la consistance relative* par rapport aux booléens. Il se trouve que l'on retrouve là des propriétés classiques dans l'étude des types abstraits. Les premiers résultats de ce travail de synthèse ont été présentés d'une part au colloque MCC-INRIA sur la résolution des équations dans des structures algébriques [20] et d'autre part au 1st Workshop on Conditional Term Rewriting Systems [35].

## 3. LANGAGES DE SPÉCIFICATIONS BASÉS SUR LA RÉÉCRITURE

(Responsables : Claude Kirchner, Hélène Kirchner; participants : Isabelle Gnaedig, Aristide Mégrelis).

Ce projet est centré autour de la conception et de la réalisation d'un langage de spécifications exécutables et de son environnement. Il comporte plusieurs aspects, théoriques et pratiques, qui se regroupent en deux thèmes :

- La réalisation d'un prototype.
- L'étude d'outils théoriques pour l'implantation : réécriture parallèle, paramétrisation, compilation.

Ce projet devrait conduire ainsi à la conception d'environnements de programmation permettant à l'utilisateur de s'assurer de la correction de ses programmes. Il s'inscrit dans le cadre de la poursuite de la collaboration qu'Hélène et Claude Kirchner ont entamée avec l'équipe de Joseph Goguen et José Meseguer lors de leur séjour au SRI en 1985-86. Une condition préliminaire à l'élaboration d'un tel projet est de partir d'un langage possédant une sémantique et un mécanisme d'inférence clairs. Le langage *OBJ* développé



[8] W. Bousdira and J.-L. Rémy, *"Reveur 4: a laboratory for conditional rewriting"*, in *Proceedings of STACS 87, Passau, Federal Republic of Germany*, pages 472-473, Passau, 1987.

[9] H. Comon and P. Lescanne, *Equational problems and disunification"*, Technical Report CRIN-88-00, Centre de Recherche en Informatique de Nancy, January 1988.

[10] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison, *"Decidability of the confluence of ground term rewriting systems"*, in *Proc. 2nd IEEE Symp. on Logic In Computer Science*, Ithaca, New York, 1987.

[11] J. Françon, B. Randrianarimanana, and R. Schott, *"Analysis of dynamic algorithms structures in d.e. knuth's model"*, in *Proc. CAAP'88*, Lecture Notes in Computer Science, 1988.

[12] J. Françon, B. Randrianarimanana, and R. Schott, *"Analysis of Dynamic data structures in D.E. Knuth's model"*, Technical Report 86-R-064, CRIN, 1986, submitted.

[13] J. Françon, B. Randrianarimanana, and R. Schott, *"Dynamic linear lists in D.E. Knuth's model: limit-distribution for the profiles of histories"*, Technical Report, CRIN, 1988.

[14] I. Gnaedig, *"Knuth-Bendix procedure and non deterministic behavior"*, *EATCS Bulletin*, 32:86-92, 1987.

[15] I. Gnaedig, C. Kirchner, and H. Kirchner, *"Equational Completion in Order-sorted Algebras"*, Technical Report 87-R-086, Centre de Recherche en Informatique de Nancy, 1987.

au SRI par l'équipe de J. Goguen et J. Meseguer, est un langage de spécification de haut niveau pour les types abstraits algébriques. *OBJ* a une sémantique algébrique basée sur les algèbres à sortes ordonnées, qui sont des algèbres dont le domaine est composé de différentes sortes avec des inclusions possibles entre elles. La théorie des algèbres à sortes ordonnées permet le polymorphisme et la surcharge des opérateurs, le traitement et la récupération des erreurs, l'héritage de propriétés et les contraintes de sortes grâce auxquelles il est possible d'exprimer certaines fonctions partielles comme des fonctions totales sur un sous-domaine défini par des équations. Les entités de base du langage sont les objets décrits par des sortes, des fonctions et des équations.

Le mécanisme d'inférence d'*OBJ* est une généralisation de la réécriture équationnelle et conditionnelle standard qui prend en compte la relation d'inclusion entre les sortes. Certaines caractéristiques du langage, telles la modularité, le traitement et la récupération d'erreurs ou les contraintes de sortes posent des problèmes théoriques délicats à résoudre.

Afin de fournir des outils de certification pour les programmes *OBJ*, et plus généralement pour des langages basés sur la logique équationnelle, il s'agit d'analyser le type de preuves qu'il est souhaitable de pouvoir faire de façon automatique. Par exemple, l'utilisateur peut spécifier que l'importation dans son programme d'un objet préalablement défini ne modifie pas le comportement de cet objet. Ce problème est lié à la preuve de théorèmes en logique équationnelle et du premier ordre pour lequel le formalisme des règles de réécriture s'avère puissant et bien adapté. Il en découle que la seconde base du projet est un démonstrateur de théorèmes en logique équationnelle et du premier ordre basé sur la complétion des systèmes de réécriture.

### *3.1. RÉALISATIONS LOGICIELLES*

• **La version 3 d'OBJ.** Le travail d'implantation d'*OBJ-3*, commencé en 85-86 au SRI, s'est poursuivi cette année, notamment au cours de deux séjours : le premier est celui de Tim Winkler du SRI pendant un mois à Nancy, le second est celui de Claude Kirchner au SRI pendant deux semaines. Ces séjours ont permis d'améliorer certains aspects du système.

• **La complétion unificationnelle en OBJ.** Une procédure de construction automatique d'algorithmes d'unification pour la classe des théories syntaxiques a été prouvée puis spécifiée en *OBJ*. Ce travail a permis par ailleurs de tester le logiciel *OBJ-3*, à la fois sur le plan de l'efficacité et sur celui du confort d'utilisation et de conception des programmes.

• **OBJ pour OBJ.** A mi-chemin entre la réalisation logicielle et l'étude théorique, un compte rendu d'expérience sur la spécification de l'interpréteur *OBJ-3* dans le langage *OBJ* a été fait : il défend la thèse qu'un travail précis de spécification est indispensable avant la programmation de gros logiciels et analyse les avantages et les inconvénients d'*OBJ* pour spécifier de tels gros programmes.

### *3.2. RÉALISATIONS THÉORIQUES*

Les réflextions théoriques sur le projet se sont concrétisées par des articles dans deux domaines :

• **Étude de la sémantique opérationnelle d'OBJ.** *OBJ* utilise un mécanisme de déclaration de sous-sortes très puissant et permettant une grande flexibilité d'expression. Il permet en particulier de traiter le problème des erreurs de façon élégante et correcte. En contre-partie, les règles de déduction deviennent plus complexes et le remplacement d'égaux par égaux n'est plus une méthode d'inférence complète. L'équivalence de deux termes doit tenir compte de la relation de sous-type. On peut trouver dans [25] une étude de la déduction dans les algèbres à sortes ordonnées, de son équivalence avec la réécriture sur des sortes ordonnées, ainsi que la description de l'interpréteur d'*OBJ-3* et des choix qui y ont été faits.

• **Étude de la complétion ordo-sortée.** Il s'avère que la complétion des systèmes de réécriture est un outil fondamental de validation de programmes écrits dans un langage basé sur la logique équationnelle.

L'utilisateur d'OBJ est supposé écrire ses programmes sous forme de systèmes de réécriture possédant la propriété de Church-Rosser. Cette hypothèse est primordiale pour que l'évaluation d'un calcul puisse se faire par réécriture et que le résultat soit unique. Si la vérification de la propriété de Church-Rosser est aisée pour de petits programmes, il n'en est pas de même pour des types de données complexes et comportant des mélanges de règles et d'équations dans lesquels l'intuition de programmeur est alors mise en défaut. Une étude systématique de la complétion de systèmes de réécriture équationnelle dans les algèbres à sortes ordonnées a été réalisée dans [15]. Les problèmes spécifiques à la logique des algèbres à sortes ordonnées sont clairement mis en lumière.

### 4. ANALYSE D'ALGORITHMES

L'étude des structures de données dynamiques dans le modèle de D.E. Knuth a été poursuivie et la réalisation pratique du système *DECIDE* est en cours [3]. Ce sera un outil d'analyse automatique d'ensembles définis par générateurs et relations et cela aura des applications concrètes en parallélisme, puisque d'après les travaux de Flé, Cori, Minoura et Roucairol on sait que le modèle sous-jacent est le monoïde partiellement commutatif. Des résultats nouveaux ont aussi été obtenus par notre groupe dans l'étude des structures dynamiques. A l'aide de techniques combinatoires, nous avions obtenu les coûts moyens de séquences d'opérations (adjonctions, suppressions, interrogations positives ou négatives) effectuées sur des fichiers représentés sous forme de listes linéaires ou de files de priorité, mais des genres plus compliqués comme les dictionnaires ou les tables des symboles ainsi que le cas où l'univers des clés est borné, demeuraient hors d'atteinte avec cette approche. Nous avons donc développé des outils algébriques permettant précisément l'analyse de ces cas, ainsi que l'étude des profils limites. Ces résultats constituent la réponse à des questions posées par D.E. Knuth [12, 13, 34, 11].

## Orientation future des recherches

Dans la suite logique des travaux théoriques menés dans l'équipe, nous nous orientons dès maintenant vers la construction d'environnements intégrés pour la preuve et la spécification de logiciels, au sein de deux projets nouveaux. L'aspect preuve correspond au projet *REVEAL*, l'aspect environnement de programmation sûre correspond au projet *TIPE*.

### 1. PROJET REVEAL

REVEAL est un projet international qui fait intervenir des chercheurs de l'Université de Stony Brook, Jieh Hsiang et Leo Bachmair, du LRI à Orsay et de l'équipe EURECA. Son but est d'étudier des systèmes de démonstration automatique essentiellement fondés sur des règles d'inférence. L'accent est mis sur les règles qui réduisent l'espace de recherche, par exemple la *simplification* par réécriture. Un autre point à examiner est la notion de *contrôle* qu'il faut ajouter aux règles d'inférence pour avoir prise sur les stratégies de preuve et sur leur efficacité lors de l'exécution. Notre programme immédiat consiste à préciser l'architecture du logiciel, les caractéristiques du langage de commande et les algorithmes de preuves dans différentes logiques.

### 2. PROJET TIPE

Le nom *TIPE* correspond aux quatre mots anglais, *Types, Inheritance, Polymorphism et Equalities*. Le but du projet est d'étudier et de réaliser un environnement *sûr* de programmation intégrant un langage de spécification puissant et exécutable à son environnement de preuve. Son objectif est de permettre le développement d'applications cri-

tiques telles qu'elles apparaissent en aéronautique ou dans les industries à haut risque (centrales nucléaires...). Le projet est en cours de définition avec des partenaires industriels belges et anglais et des universitaires français et allemands. Il fera l'objet d'une soumission au programme européen ESPRIT-2.

## 3. ASPECTS FONDAMENTAUX

Les travaux théoriques liés à nos quatre domaines de recherche actuels s'intègrent à la recherche fondamentale qui sous-tend ces deux projets. Chaque pôle poursuit des objectifs très précis.

### *3.1. RÉÉCRITURE ET DÉDUCTION AUTOMATIQUE*

Sur la terminaison, l'étude des systèmes de réécriture associative et commutative nécessite d'être poursuivie pour être mieux comprise. Sur la méthode des valuations polynômiales, nous comptons créer des systèmes qui aident l'utilisateur à trouver les valuations, l'un des points délicats de la méthode. D'autre part, puisque la méthode par transformation est fondée sur une propriété proche de la confluence, nous voulons développer une procédure qui fonctionnerait comme une complétion et qui offrirait automatiquement les différents paramètres de la preuve de terminaison.
Dans le domaine prometteur de l'étude de la désunification, domaine dans lequel seuls les premiers pas ont été faits, nos recherches porteront sur ses liens avec la surréduction.

### *3.2. LA RÉÉCRITURE CONDITIONNELLE*

Les perspectives concernant la réécriture conditionnelle ont été dégagées pour l'essentiel à l'intérieur de la présentation précédente. Nous comptons poursuivre la recherche théorique sur les systèmes hiérarchiques, développer une implantation sur des bases nouvelles, en prenant en particulier en compte la description des algorithmes par des règles d'inférence, et progresser dans la synthèse entre les différentes approches. Ces différents points s'intègrent dans le travail de thèse de doctorat de Wadoud Bousdira, chercheuse associée au projet.

### *3.3. LANGAGES DE SPÉCIFICATIONS BASÉS SUR LA RÉÉCRITURE*

L'expérience acquise grâce à l'étude approfondie du langage *OBJ* et de ses mécanismes permet de mieux appréhender les problèmes qui sont au centre du projet TIPE. Continuant dans cette voie, nos projets sont l'étude, la généralisation et l'implantation d'outils permettant de valider des programmes, c'est-à-dire de montrer l'unicité de leurs résultats et leur terminaison. Ces outils tiendront compte de la structuration des programmes permise par les mécanismes d'enrichissement et de paramétrisation.

### *3.4. ANALYSE D'ALGORITHME*

**Les recherches vont s'orienter vers l'analyse :**

- d'algorithmes liés à la réécriture (filtrage, unification, surréduction), par des techniques de fonctions analytiques (Méthode du col, transformée de Mellin, etc.),
- de systèmes de réécriture (des résultats partiels ont été obtenus par P. Lescanne, C. Choppy, S. Kaplan, J.L. Remy et M. Soria) et vers l'introduction d'outils probabilistes sophistiqués pour affiner l'étude des structures de données dynamiques dans le modèle de Knuth. D'autre part, H. Amet a entrepris un travail portant sur la conception et l'analyse de processus décisionnels pour les déplacements dans $\mathbf{R}^2$ et $\mathbf{R}^3$.




[16] I. Gnaedig and P. Lescanne, *"Proving termination of associative rewriting systems by rewriting",* in J. Siekmann, editor, *Proc. 8th Conference on Automated Deduction, Oxford,* Springer Verlag, Lecture Notes in Computer Science, Oxford (England), 1986.

[17] J. Hsiang and M. Rusinowitch, *"On word problems in equational theories",* in *Int. Coll. on Automata Languages and Programming,* 1987.

[18] J. Hsiang, M. Rusinowitch, and K. Sakai, *"Complete inference rules for the cancellation laws",* in *Int. Joint Conf. on Artificial Intelligence,* 1987.

[19] J.-P. Jouannaud and P. Lescanne, *"La réécriture", Techniques et Sciences Informatiques,* 5(6), 1987.

[20] S. Kaplan and J.-L. Remy, *"Completion algorithms for conditional rewriting systems",* in *Proc. Colloquium on Resolution of Equations in Algebraic Structures,* MCC, 3500 West Balconies Center Drive, Austin, Texas 78759-6509, May 4-6 1987.

[21] C. Kirchner, *"Computing unification algorithms",* in *Proceeding of the first symposium on Logic In Computer Science, Boston (USA),* pages 206-216, 1986.

[22] C. Kirchner, *"From Unification in Combination of Equational to A New AC-Unification algorithm",* Technical Report 87-R-132, Centre de Recherche en Informatique de Nancy, 1987. To appear in the proc. of the CREAS.

[23] C. Kirchner, *"Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles"*, Thèse d'État de l'Université de Nancy I, 1985.

[24] C. Kirchner, editor, *"Proceedings on a workshop on Unification"*, Centre de Recherche en Informatique de Nancy, BP 239, 54506 Vandœuvre-lès-Nancy, France, 1987.

[25] C. Kirchner, H. Kirchner, and J. Meseguer, *"Operational semantics of OBJ3"*, Technical Report 87-R-87, Centre de Recherche en Informatique de Nancy, 1987.

[26] C. Kirchner and P. Lescanne, *"Solving disequations"*, in *Proc. IEEE 2nd Symp. on Logic in Computer Science*, 1987.

[27] H. Kirchner, *"Schematization of infinite sets of rewrite rules, application to the divergence of completion processes"*, in P. Lescanne, editor, *Rewriting Techniques and Applications*, pages 180-191, Springer Verlag, 1987.

[28] A. Lazrek, P. Lescanne, and J.-J. Thiel, *"Proving inductive equalities, Algorithms and Implementation"*, Technical Report 682, INRIA, Centre de Recherche en Informatique de Nancy, 1986.

[29] P. Lescanne, *"Current Trends in Rewriting Techniques and Related Problems"*, Technical Report CRIN 87-R-051, Centre de Recherche en Informatique de Nancy, 1987. Invited Conference at IBM Symposium on *Current Trends in Computer Algebra.*

[30] P. Lescanne, *"Divergence of the Knuth-Bendix procedure and termination orderings"*, *Bull. EATCS*, 30:80-83, October 1986.

[31] P. Marchand and R. Schott, *"Résolution par algorithmes de problèmes dans les groupes libres"*, Technical Report 86-R-066, CRIN, 1986.

[32] J. Mzali, *"Méthodes de filtrage équationnel et de preuve automatique de théorèmes"*, PhD thesis, Université de Nancy I, September 1986.

[33] W. Nutt, P. Rety, and G. Smolka, *"Basic Narrowing Revisited"*, Technical Report SR-87-07, Universität Kaiserslautern, West Germany, July 1987.

[34] B. Randrianarimanana, *"Analyses des structures de données dynamiques dans le modèle de D.E. Knuth"*, Thèse de 3° cycle, Université de Nancy I, 1986.

[35] J.-L. Rémy, *"Topics on conditional term rewriting systems"*, in *1er Workshop on Conditional Term Rewriting Systems*, Orsay, 1987.

[36] P. Réty, *"Étude de la surréduction"*, Thèse de l'Université de Nancy I, March 1988.

[37] P. Réty, *"Improving basic narrowing techniques and commutation properties"*, in *2nd International Conference on Rewriting Techniques and Application*, 1987.

[38] P. Réty, C. Kirchner, H. Kirchner, and P. Lescanne, *"An algorithm for unification based on narrowing"*, Technical Report 86-R-131, Centre de Recherche en Informatique de Nancy, 1986.

[39] J. Rouyer, *"Preuves de terminaison fondée sur les interprétations polynômiales : implantation d'une méthode basée sur le théorème de Sturm"*, Rapport de DEA, Université de Nancy I, 1987.

[40] M. Rusinowitch, *"Démonstration automatique par des techniques de réécriture"*, PhD thesis, Université de Nancy I, 1987.

# Théorie et utilisation des grammaires attribuées

ATTRISEM *MOTS CLES*
Ada (langage de programmation)
attributs sémantiques
environnements de programmation
grammaires attribuées
grammaires d'arbres
programmation logique
prolog
sémantique dénotationnelle
sémantique des langages
spécifications formelles
types abstraits algébriques

Le rapport ALGOL60 révisé en 1962 définissait informellement, c'est-à-dire en langage courant, la *Sémantique* du langage, c'est-à-dire la fonction calculée par un programme supposé syntaxiquement correct. Comme D. Knuth l'a bien montré, cette définition laissait de nombreuses incertitudes et ambiguïtés dont la résolution revenait *de facto* à la personne chargée d'écrire le compilateur ou l'interprète du langage. Le langage y perd ainsi son caractère *d'indépendance de la machine qui l'implante.* Cette indépendance devait faire du langage ALGOL, un langage de référence pour la définition d'algorithmes, elle était donc essentielle.

Le nécessité de définir formellement la sémantique des langages de programmation s'est donc imposée très tôt. "Formellement" veut dire par exemple au moyen d'un ensemble des déquations associées aux règles de syntaxe qui spécifie de manière unique la fonction calculée par le programme, et de préférence de façon opératoire, calculable par ordinateur. Une telle définition débouche ainsi sur la réalisation d'un méta-interprète, prenant en entrée la définition du langage et un programme, et fournissant le résultat.

Comme B. Lorho l'a clairement exposé dans sa thèse, les définitions formelles des langages de programmation concernent :
1. l'écrivain du compilateur ou de l'interprète : la définition formelle du langage lui sert alors de référence, de spécification,
2. le programmeur qui pourra l'utiliser pour vérifier la validité de son programme, ou en tout cas pour le mettre au point plus rapidement que par une suite d'essais et de corrections des erreurs,
3. le théoricien qui cherche à comparer différents langages du point de vue de leur efficacité, de leur facilité d'utilisation ou à définir des méthodologies de programmation.

Différents formalismes ont été proposés. Nous n'en citerons que deux.

La *"sémantique dénotationnelle"* est un formalisme proche du lambda-calcul qui fait intervenir des constructions de domaines complexes. Cette technique a été utilisée pour mettre au point et définir le langage ADA.

La seconde est la méthode des *"attributs sémantiques"* qui est l'objet des travaux du groupe ATTRISEM. Elle a été proposée par D. Knuth dans un article publié en 1968. Elle est plus directement apte à l'implantation car ses règles sémantiques sont des affectations d'un langage à affectation unique (c'est-à-dire où chaque variable ne reçoit qu'une valeur au cours de l'éxécution d'un programme).

Cette méthode est *déclarative* en ce sens qu'elle définit des objets sans donner exactement la méthode de calcul. (L'existence d'un algorithme est assurée par un test, dit de *non-circularité* qui doit être fait après l'écriture des règles sémantiques).

De nombreuses implantations ont été réalisées dans le monde, toutes orientées vers l'utilisation des grammaires attribuées pour la compilation et l'interprétation des langages de programmation. Ces implantations définissent des stratégies d'évaluation et cherchent les meilleurs compromis possibles entre des impératifs contradictoires : temps de calcul, espace-mémoire utilisé, types de grammaires attribuées acceptables (on ne connaît pas d'algorithme efficace qui accepte toutes les grammaires attribuées non circulaires).

Les grammaires attribuées ont également fait l'objet de recherches théoriques approfondies : complexité des algorithmes d'évaluation, type de relation de traduction définie, comparaisons avec les transductions d'arbres, etc.

Il faut enfin mentionner la parenté avec d'autres formalismes tels que PROLOG, qui fait elle aussi le sujet de recherches prometteuses.

Dès la création du GRÉCO PROGRAMMATION, le groupe ATTRISEM s'est constitué pour étudier les différents aspects théoriques et pratiques des grammaires attribuées et a retenu les thèmes majeurs suivants :

1. Algorithmes d'évaluation et implantation des grammaires attribuées : étude bibliographique systématique [DJL 88], système FNC-2 [Fil 87, Par 87, JP 88] d'aide à la compilation. Des extensions sont prévues avec évaluation incrémentale des attributs.

2. Applications des grammaires attribuées : environnement de programmation et éditeurs graphiques [AF 87, Zar 86, Lev 88], compilation de TYPOL, langage de spécifications mis au point dans le projet INRIA-CROAP [Att 88].

3. Relations entre les grammaires attribuées et d'autres formalismes tels que la programmation en logique [DM 85, CD 87] ou les grammaires d'arbres [Bar 87]. Ces travaux ont d'abord un caractère théorique mais conduisent à des applications dans le domaine des spécifications formelles et de leur exécution [Att 88].

## Logiciels "industriels" réalisés ou en cours de réalisation

FNC-2 : spécification et évaluation de grammaires attribuées.

GRAPHADA : un éditeur syntaxique graphique pour ADA.

ADA : un préprocesseur orienté objets pour ADA.



## Références


[Zar 86] A. Zarli, *"Édition structurée : application à un éditeur de texte guidé par la syntaxe pour le langage ADA"*, journées ADA/AFCET ENST, Paris, 1986.

[AF 87] I. Attali, P. Franchi-Zannettaci, *"Inference system environment for ADA"*, conf. ADA-EUROPE, Stockholm, 1987.

[Att 88] I. Attali, *"Compiling out unification in TYPOL"*, conf. INFORMATIKA, Nice, 1988.

[Lev 88] D. Levi, *"Problèmes de réutilisation liés au typage, application à une extension du langage ADA"*, thèse de doctorat, Université de Nice, 1988.

[DJL 88] P. Deransart, M. Jourdan, B. Lorho, *"A survey on Attribute Grammars"*, INRIA RR 417, 485, 570 mis à jour et à paraître chez Springer Verlag.

[CD 87] B. Courcelle, P. Deransart, *"Proof of Partial Correctness of Attribute Grammars with Application to Recursive Procedure and Logic Programming"*, Université de Bordeaux, RR18702, à paraître dans Information and Computation.

[DM 85] P. Deransart, J. Maluszynski, *"Relating Logic Programs and Attribute Grammars"*, J. of Logic Programming 1985, 2 pp 119-155.

[Bar 87] K. Barbar, *"Attributed Context-Free Tree Grammars"*, Université de Bordeaux I, RR 8716, avril 1987.

[Fil 87] G. Filé, *"Classical and Incremental Evaluators for Attribute Grammars"*, TCS 53,1, 1987, pp 25-65.

***Responsable scientifique***
P. Deransart

***Organismes***
I.N.R.I.A. Rocquencourt
Domaine de Voluceau Rocquencourt
78150 Le Chesnay
☎ (1) 39 63 55 11
I.N.R.I.A. Sophia-Antipolis
Université d'Orléans
Université de Strasbourg

***Participants***
GROUPE ATTRISEM
P. Deransart
M. Jourdan
D. Parigot
B. Lorho,
C. Julié
P. Franchi-Zannettacci
A. Zarli
C. Kleinhans
M. Fornarino
B. Chabrier
J.-P. Forestier
J.-P. Giacometti
I. Attali
D. Lévi
K. Barbar
G. Filé
J.-M. Schramm

ADAGE : un analyseur syntaxique compatible YACC, attribué, en ADA, générant du code ADA.

PROLOG-ADA : un interprète PROLOG à stratégies paramétrables écrit en ADA, interfaçable avec des packages ADA.

## Contacts Extérieurs

Universités de Cornell, Brandeis, Philadelphie (U.S.A.), Karlsruhe, Dortmund, Saarbrücken (R.F.A.), Rostock, Berlin-Est (RDA), Moscou, Tallin (URSS), Linköpirg, Göteborg (Suède), Helsinki (Finlande), Tokyo (Japon), Rio de Janeiro-Federale, Recife (Brésil), Braga, Lisbonne (Portugal), Imperial College (Londres), Padoue (Italie).

## Contacts Industriels

Xerox, CR2A, Alsys, Ilog, Paris, IBM La Gaude, Thomson SINTRA Cagnes-sur-Mer, CISI Sophia Antipolis, Bull, ICL (UK).

## Projet ESPRIT n° 956 "COCOS"

Le projet COCOS (Components for Future Computing Systems) étudie une architecture ouverte pour une station de travail à usage bureautique, ainsi que certains composants matériels et logiciels s'intégrant à cette architecture. Les principaux thèmes de recherche dans COCOS sont les suivants :

- implantation du langage de programmation logique parallèle PARLOG sur une base matérielle associant uSyC (microprogrammable symbolic coprocessor) à un CPU classique (68020 ou ARM, Acorn RISC Machine) ; le compilateur de PARLOG en code intermédiaire SPM est réalisé à l'aide du système FNC-2 ;
- développement de nouveaux concepts de programmation, en particulier la programmation orientée objet, sur PARLOG ;
- développement d'une architecture, d'outils génériques et d'outils spécialisés pour l'interface homme-machine ;
- développement et implantation du langage orienté objet fortement typé Le-Tool.

La contribution de l'Équipe "Langages et Traducteurs" de l'INRIA au projet COCOS porte sur les deux premiers points. Publication : The FNC2 System : Advances in Attribute Grammars Technology.


[JP 88] M. Jourdan, D. Parigot, *"The FNC-2 System, User's Guide and Reference Manual"*, INRIA, février 1988.

[Par 87] D. Parigot, *"Mise en œuvre des Grammaires Attribuées : Transformation, Évaluation Incrémentale, Optimisations"*, thèse de 3e cycle, Université de Paris-Sud, Orsay, juin 1987.

# Un langage de programmation en logique Modale

**MODAL** *MOTS CLES*
compréhension du langage naturel
démonstration automatique
environnements de programmation
formulation du raisonnement
intelligence artificielle
logique des situations
logique modale
logique temporelle et parallélisme
machine abstraite/virtuelle
MOLOG (langage de programmation logique)
stratégie de preuve

Suite au premier travail sur la logique des situations de McCarthy et Hayes, un grand nombre de recherches, utilisant la logique modale, ont été développées en Intelligence Artificielle dans les domaines suivants : Compréhension du langage naturel, Formalisation du raisonnement (raisonnement sur des données incomplètes ou incertaines, raisonnement temporel, raisonnement sur la connaissance), et dans une autre "tradition", la Logique des programmes.

L'utilisation de la logique modale en informatique pose le problème de l'automatisation de la déduction dans ces formalismes. Plusieurs méthodes existent : *traduction* en logique du premier ordre classique, *déduction naturelle* et *résolution*.

C'est ce dernier type de méthode que notre projet étudie tout particulièrement.
Cette méthode consiste à étendre la résolution classique à la logique modale. L'intérêt fondamental réside dans la possibilité de définir des stratégies de preuve, un point essentiel en démonstration automatique.
Un développement important de notre approche a été de définir une notion de "clause modale de Horn", par analogie avec les clauses du même nom en logique classique sur lesquelles, on le sait, est construit le langage Prolog. Un langage de programmation "en logique modale", MOLOG, en est issu.

Les objectifs du projet sont :

- Développer les bases logiques de la méthode de résolution.
- Développer MOLOG comme langage de programmation : implémentation, sémantique...
- Étudier – en s'appuyant en particulier sur ce langage – les applications de la logique modale.

## Principaux résultats obtenus en 86 et 87

Les travaux se sont poursuivis dans les trois directions suivantes :

### 1. ÉTABLISSEMENT DES BASES LOGIQUES DE LA RÉSOLUTION

- *RÉSOLUTION POUR LA LOGIQUE PROPOSITIONNELLE :*

En utilisant une technique basée dans la méthode de Skolem, nous avons pu définir des systèmes modaux (propositionnels) pour lesquels, d'une part, on peut donner des règles de résolution relativement simples et, d'autre part, simplifier les preuves de complétude des règles.

- *STRATÉGIES DE RÉSOLUTION :*

Un ensemble de stratégies classiques (linéaire, négative, input...), permettant des gains en efficacité, a été obtenu dans le cadre de la logique modale "classique" ou de ces extensions.

- *RÉSOLUTION POUR LE 1^er^ ORDRE :*

a) Une propriété de Herbrand pour le système modal Q (le plus simple des systèmes modaux) a été établie, ce qui permet d'étendre la résolution au premier ordre. Cette méthode a été développée dans la thèse d'Université que Marta Cialdea a soutenue à l'Université P.-Sabatier (Toulouse).

Cette méthode a été implémentée sous forme d'un démonstrateur utilisant la méthode du graphe de Kowalski. La définition et la mise en place de stratégies sont en cours.

b) Comme pour le calcul propositionnel, nous avons étendu aussi les techniques de Skolem pour la logique modale de premier ordre, et à nouveau, certains problèmes difficiles en déduction automatique, cette fois-ci liés à la quantification, se simplifient. Nous avons appliqué cette technique à plusieurs systèmes modaux et les premiers résultats sont encourageants puisque nous traduisons les problèmes de déduction automatique en logique modale par des problèmes de déduction dans des théories équationnelles.

### 2. AUTOUR DE MOLOG

- *UNE MACHINE ABSTRAITE POUR MOLOG.*

Nous avons défini une méthode pour compiler des clauses MOLOG. Pour cela, un ensemble de primitives de programmation a été défini, qui étend celles de la "Machine de Warren". La méthode de traduction présentée produit, à partir d'une clause MOLOG, un programme écrit à l'aide de ces primitives. Le but de ce travail est d'obtenir des implémentations efficaces pour les logiques modales.
L'implémentation - actuellement en lisp - commence à être refaite en C pour des raisons d'efficacité évidentes.

- *SÉMANTIQUE DÉCLARATIVE POUR MOLOG.*

Une sémantique déclarative pour certains systèmes de logique modale a été définie. En faisant correspondre à chaque clause modale de Horn une transformation continue comme dans le cas de clauses de Horn classiques, on peut associer à chaque programme un modèle minimal. On obtient une méthode pour prouver la complétude des clauses de Horn modales de plusieurs systèmes, Q, T, S4, et une compréhension informatique des clauses modales de Horn.

### 3. APPLICATIONS

- *RAISONNEMENT HYPOTHÉTIQUE*

Nous avons élaboré un système de logique modale permettant de représenter certains aspects liés à la mise à jour de connaissances dans une base de données déductive. Pour cela, nous traduisons la mise à jour par l'introduction d'une nouvelle hypothèse ; faire



## Références

Y. Aufray, *"Linear strategy for modal resolution"*, à paraître Information Proc. Letters.

Y. Aufray, P. Enjalbert *"Modal Theorem proving: an equationnal approach"*. Journées Européennes sur les méthodes logiques en I.A. 1988, Roskoff.

Y. Aufray, P. Enjalbert, J.-J. Hébrard, *"Strategies for modal resolution: results and problems"*, rapport L.I.U.C. 87-2, université de Caen, septembre 1987.

R. Arthaud, P. Bieber, L. Fariñas Del Cerro, J. Henry, A. Herzig, *"Automated modal reasoning, Proceedings of the Int. Conf. on Information Processing and Management of Uncertainly in knowledge Based Systems"*, Paris, Juillet 1986.

R. Arthaud, P. Bieber, L. Fariñas Del Cerro, L. Henry, A. Herzig, MOLOG : *"Manuel d'utilisation"*, rapport L.S.I., février 1986.

E. Audureau, P. Enjalbert, L. Fariñas Del Cerro, *"Théorie de la programmation et logique temporelle. 1^re^ partie : Méthodes de preuve T.S.I., 6, 6,* 1987, *2^e^ partie : Validation d'algorithmes parallèles", T.S.I.*, à paraître, 1988.

E. Audureau, P. Enjalbert, L. Fariñas Del Cerro, *"Logique Temporelle : Sémantique et validation de programmes parallèles"*, ouvrage à paraître chez Masson, 1988.

S. Badaloni, Cl. Sayettat, *"L'Aspect Temporel en Intelligence Artificielle"*, Intelligence Artificielle et CAO en BTP, Ed. Hermès 1988.

— Caen
— Toulouse
— Nantes


***Responsable scientifique***
Patrice Enjalbert
Luis Fariñas Del Cerro

***Organismes***
CER de Mathématiques
Esplanade de la Paix
Université de Caen
14032 Caen Cedex
☎ 31948140

L.S.I. Toulouse
Université Paul-Sabatier
118, route de Narbonne
31077 Toulouse
☎ 61556763

L.I.S.I. - E.N.S.M.
1, rue de la Noë
44072 Nantes cedex 03
☎ 40371600

***Participants***
Caen
P. Enjalbert
J.-J. Hébrard
Y. Auffray
C. Bastin

Toulouse
L. Fariñas Del Cerro
P. Bieber
A. Herzig
J.-M. Alliot
C. Seguin
J. Garmendia-Aguirre
C. Sayettat
C. Bouchy
A. Potet


une requête à la base de données après avoir effectué une mise à jour revient à réaliser un raisonnement hypothétique. En d'autres termes, la base de données n'est pas modifiée réellement par la mise à jour, ce qui nous permet de raisonner sur les mises à jour elles-mêmes.

• *EXPRESSIVITÉ EN LOGIQUE MODALE*
Dans le cadre d'une étude entreprise sur le pouvoir expressif de la logique modale, il a été montré que pour chaque grammaire formelle G, il peut être défini une logique modale propositionnelle LG vérifiant la propriété suivante :
si un mot m est engendré par la grammaire G, alors il existe une forme $F_m$ associée à m telle que $F_m$ est un théorème de LG.
Une conséquence immédiate de ce travail est de disposer d'une méthode simple pour construire des logiques modales propositionnelles indécidables. Le but de cette étude est d'étendre les grammaires logiques afin de traiter dans un même cadre formel des aspects syntaxiques et sémantiques liés à la compréhension du langage naturel.

• *LOGIQUE TEMPORELLE ET VALIDATION DE PROGRAMMES PARALLÈLES*
Enfin, nous avons achevé un travail de synthèse sur les applications de la logique temporelle (une des applications-variantes de la logique modale) à la validation de programmes parallèles. De ce travail résultent deux articles parus dans T.S.I. et un livre à paraître en 1988.

• *ASPECT TEMPOREL EN INTELLIGENCE ARTIFICIELLE*
Afin de dégager une synthèse des travaux réalisés autour ou à partir des logiques temporelles, nous avons travaillé dans plusieurs directions. Tout d'abord la réalisation d'un démonstrateur basé sur la méthode des tableaux sémantiques pour diverses logiques propositionnelles temporelles, ainsi qu'une nouvelle implantation de MOLOG en Prolog II. D'autre part, nous avons mis en place l'ensemble des opérateurs temporels des logiques ITL, LTTL, et de Allen afin de disposer d'un outil de test de ces différentes approches, une application à la gestion des processus concurrents a été faite.

• *RAISONNEMENT DANS UN UNIVERS MULTI-AGENTS*
Un tel univers peut se représenter par un modèle multi-modal associant le temps (pour représenter l'évolution) et les connaissances des différents agents. Un résolveur de problèmes sur cet univers est en cours d'étude et l'approche retenue est la mise en place de stratégies de déduction sur la méthode des tableaux sémantiques.

## Collaborations industrielles

### PROJET ALPES (P973 DU PROGRAMME ESPRIT)

Le thème de recherche du projet Alpes (Advanced Logic Programming EnvironmentS) consiste dans la définition et l'implémentation d'un environnement de programmation pour Prolog. ALPES possède trois niveaux :

• un niveau central (lié au compilateur) dans lequel on définit des primitives permettant de définir un système graphique et de modifier l'algorithme d'unification,

• un niveau dynamique (lié à l'exécution du programme) dans lequel on définit des outils pour la modification du contrôle de l'exécution des programmes,

• un niveau statique, auquel se rattache la recherche de notre équipe, dans lequel on définit des primitives permettant le contrôle de types, la mise au point de programmes, etc.

Ce projet est réalisé en collaboration avec l'Université de Lisbonne, la société Conception et Réalisation Industrielles de Logiciel (CRIL), la société BULL, ENIDATA (Bologne, Italie), Technische Universität München (TUM), le Laboratoire de Recherche en Informatique (Orsay).


Ph. Balbiani, L. Fariñas Del Cerro, A. Herzig, *"Declarative semantics for modal logic programs"*, rapport L.S.I., novembre 1987.

P. Bieber, *"Programmation modale et représentation des connaissances"*, rapport de DEA, UPS, Toulouse, septembre 1986.

C. Bouchy, *"Résolution Modale pour la Logique du 1er Ordre"*, rapport LISI 1987.

M. Cialdea, *"Un metodo di rizoluzione per il calcolo dei predicati modale"*, Primo convegno sulla Programmazione Logica, Genova, 12-14 mars 1986.

M. Cialdea, *"Une méthode de déduction automatique en logique modale"*, thèse de doctorat de l'Université Paul Sabatier, Toulouse, 1986.

P.M. Delpech, A. Potet, Cl. Sayettat, *"Démonstration Automatique et Logique Temporelle"*, TSI, Vol. 6, n° 6, 1988.

R. Demolombe, L. Fariñas Del Cerro, *"Representation on incomplete information about structured objects"*, rapport interne CERT-L.S.I., 1987.

R. Demolombe, L. Fariñas Del Cerro, *"An algebraic evaluation method for deduction in incomplete database"*, The journal of Logic Programming, à paraître, 1988.

P. Enjalbert, L. Fariñas Del Cerro, *"Modal resolution in clausal form"*, Theoretical Computer Science, à paraître.

M. Cialdea, L. Fariñas Del Cerro, *"A Modal Herbrand Property, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik"*, 32, 1986, pp 523-530.

L. Fariñas Del Cerro, *"MOLOG, a system that extends PROLOG with modal logic"*, New Generation Computing, 4, 1986, pp 35-50.

L. Fariñas Del Cerro, A. Herzig, *"Reasoning about data base updates"*, Workshop on foundations of Deductive databases and logic programming, Washington, août 1986, pp 53-67.

L. Fariñas Del Cerro, A. Herzig, *"Linear modal deductions"*, rapport L.S.I., UPS, novembre 1987.

L. Fariñas Del Cerro, A. Herzig, *"An automated modal logic of elementary changes"*, in Non-standard logics for automated reasoning, P. Smets, A. Mandani, D. Dubois, H. Prade (Eds.), North Holland, à paraître.

L. Fariñas Del Cerro, A. Herzig, *"Quantified modal logic and unification theory"*, rapport L.S.I., UPS, janvier 1988.

L. Fariñas Del Cerro, M. Penttonen, *"A note on the complexity of the satisfiability of modal Horn clauses"*, The Journal of Logic Programming, 4, 1987, pp 1-10.

L. Fariñas Del Cerro, M. Penttonen, *"A simple method to construct undecidable propositional modal logics"*, rapport Université de Turku (Finlande), 1987.

O. Nerrand, Cl. Sayettat, *"Prolog et les Opérateurs Temporels*, 7° Séminaire de Programmation Logique, Trégastel 1988.

A. Potet, Cl Sayettat, *"Application des Logiques Modales et Temporelles à la Représentation du Raisonnement dans un Environnement Multi-agents"*, Journées Européennes sur les méthodes Logiques en Intelligence Artificielle (JELIA), Roscoff 1988.

# Langage et Systèmes de Calcul Formel

Ce projet est commun à L'INRIA et à l'École Normale Supérieure

ÉQUIPE INRIA
***Responsable scientifique***
Gérard Huet

ÉQUIPE LIENS
***Responsable scientifique***
Guy Cousineau

FORMEL *MOTS CLES*
calcul des constructions
calcul symbolique
CAML (langage fonctionnel)
démonstration automatique
environnement de preuve
environnement de programmation
évaluation paresseuse
génie logiciel
intelligence artificielle
lambda-calcul
langage fonctionnel
logique linéaire
machine virtuelle
types

Le projet Formel a pour but de concevoir et développer des systèmes de calcul formel et de les appliquer aux outils de base de génie logiciel. Ce projet est formé d'une équipe INRIA à Rocquencourt et d'une équipe du Laboratoire d'Informatique de l'École Normale Supérieure (LIENS) travaillant en étroite collaboration. Cette collaboration a été officialisée cette année par la signature d'une convention entre les deux organismes.
Notre projet a pour ambition de développer les concepts et les outils de base nécessaires à la tâche de conception et de mise au point d'un langage de programmation selon tous ses aspects de syntaxe, sémantique et déduction.
Une tâche essentielle est de concevoir un système formel puissant capable de manipuler des axiomatisations, spécifiant par exemple les propriétés algébriques que doivent vérifier les opérateurs du langage et des définitions, comme par exemple les programmes eux-mêmes, les règles de formation et de typage de ces programmes, les interpréteurs et compilateurs, les règles d'inférence de la logique. De plus, un tel système doit pouvoir être utilisé comme base d'un environnement de programmation pour le langage. C'est notre thèse qu'un environnement de programmation ambitieux doit s'appuyer sur une machine très générale de manipulation de théories mathématiques. Notre projet s'inscrit donc dans un programme à plus long terme d'Informatique Fondamentale, vue comme cadre d'Investigation des Mathématiques Constructives.

L'unité de nos préoccupations est assurée par l'existence d'une théorie noyau, le lambda-calcul, qui se trouve être un outil central aussi bien en Logique Mathématique (Théorie de la Démonstration) qu'en Informatique (Langages Fonctionnels - Sémantique Dénotationnelle). Nos recherches s'orientent selon deux axes principaux : l'implantation des langages fonctionnels et la conception d'environnements de preuves.
Nous décrivons ici sans souci d'exhaustivité quelques-unes de nos actions de recherche actuelles.

## Actions de recherche

### 1 LANGAGES FONCTIONNELS

Une part importante de l'activité de notre projet concerne l'implémentation du langage CAML. Il s'agit d'un langage à la fois interactif, compilé (au vol) et doté d'une structure de types génériques permettant la synthèse automatique des types par le compilateur. Ses domaines d'application sont le Génie Logiciel et l'Intelligence Artificielle. Cette activité de développement peut être vue comme application et en même temps source d'inspiration pour les études théoriques sur les langages fonctionnels que nous menons par ailleurs.

*1.1 DÉVELOPPEMENT DE CAML*

Notre équipe a poursuivi son effort de conception et d'implantation du langage CAML. Cet effort a abouti à la diffusion à l'extérieur de la version 2.5 du langage. Cette diffusion, effectuée par les membres du projet pour les versions en beta-test, fait maintenant l'objet d'un accord avec ILOG, qui assure la distribution, le premier niveau de maintenance, et la promotion industrielle.
Le langage CAML a acquis le statut d'un logiciel permettant le développement de gros systèmes de calcul symboliques grâce aux efforts qui ont porté sur les outils de manipulation de syntaxes (analyseur lexical en CAML, grammaires de langages-objets, générateur d'automates), sur les outils de production de documents (Interface Latex réalisée dans l'esprit du système WEB) et sur la documentation (Écriture d'un Primer et d'un Manuel de Référence). Cet effort de développement est mené principalement par M. Mauny et P. Weis avec la collaboration de V. Aponte, A. Laville et D. Rémy.
De nombreux travaux sont en cours pour améliorer l'interface système de CAML (interface avec C et X-windows) ainsi que son environnement de programmation (édition structurée, débogage symbolique, compilation séparée) et pour étendre le langage vers la programmation logique et la programmation orientée objets.

*1.2 THÉORIE DES LANGAGES FONCTIONNELS*

Les langages fonctionnels se prêtent particulièrement bien à la description formelle pour leur typage ou leur sémantique et nous avons pu pousser très loin cette formalisation avec la machine catégorique (CAM) décrite par G. Cousineau et P.-L. Curien. Elle permet de décrire de façon fine un mécanisme de production de code machine pour les programmes fonctionnels. C'est sur cette machine virtuelle qu'est bâtie l'implémentation de CAML.

Cette approche a trouvé son prolongement dans les travaux d'Yves Lafont sur la machine linéaire. Celle-ci s'appuie sur la Logique Linéaire de Jean-Yves Girard dans sa version intuitionniste et présente par rapport à la CAM l'intérêt supplémentaire de prendre en compte d'un point de vue théorique les problèmes d'allocation mémoire. D'une façon plus générale, il semble bien que la Logique Linéaire soit porteuse d'applications informatiques qui vont bien au-delà des langages fonctionnels et qu'elles permettent de rendre compte de phénomènes tels que le parallélisme et la bi-directionnalité des transferts d'information. Cette orientation est actuellement explorée par Y. Lafont et V. Danos.

Ulm

| **Organisme** | **Participants** | **Organisme** | **Participants** | |
|---|---|---|---|---|
| École Normale Supérieure<br>Département de Mathématiques et Informatique<br>45, rue d'Ulm<br>75005 Paris<br>(1)43291225 | Thierry Coquand<br>Philippe Le Chenadec<br>Michel Mauny<br>Pierre Weis<br>Maria-Virginia Aponte<br>Annie Forêt<br>Thérèse Hardin<br>Alain Laville<br>Didier Rémy | INRIA (Rocquencourt)<br>Domaine de Voluceau<br>Rocquencourt<br>78150 Le Chesnay<br>(1)39635511 | Pierre-Louis Curien<br>François Fages<br>Laurent Fribourg<br>Yves Lafont<br>Christine Mohring-Paulin<br>Ramon Pino | Laurence Puel<br>Anne Bouverot<br>Loïc Colson<br>Vincent Danos<br>Catherine Delor<br>Thomas Ehrhard<br>Vincent Poirriez |

Du point de vue du typage, nous étudions différentes possibilités pour introduire la notion d'inclusion de types à l'intérieur du polymorphisme afin de pouvoir prendre en compte la notion d'héritage des langages orientés objets.
Enfin, un effort particulier a été porté sur les problèmes liés à l'évaluation paresseuse des langages fonctionnels et en particulier sur la possibilité d'effectuer le filtrage de façon paresseuse. A. Laville a obtenu des résultats qui font progresser ce domaine de façon significative.

## 2 ENVIRONNEMENTS DE SPÉCIFICATION ET DE PREUVES

L'outil méthodologique fondamental pour ce thème est une correspondance (dite de Curry-Howard), qui relie la notion de type (resp. programme) à la notion de proposition (resp. preuve). Ceci permet de guider la conception de types d'un langage de programmation suivant des critères logiques permettant leur utilisation en tant qu'assertion. Un programme bien typé est alors un programme correct vis-à-vis de ses spécifications. De plus, on peut voir le programme lui-même comme une traduction de la preuve de cohérence des types ce qui permet d'envisager de faire de la synthèse de programme par des méthodes de démonstration automatique.

### *2.1 LE CALCUL DES CONSTRUCTIONS*

Les recherches sur la conception d'environnements de preuve se sont poursuivies autour du Calcul des Constructions proposé par Th. Coquand et G. Huet. Une version CAML plus efficace a été implantée par G. Huet, autour d'un noyau de "Machine Constructive" particulièrement compact. Cette machine manipule des environnements de types et valeurs interdépendants, permettant de construire de manière homogène des types, des propositions logiques, des preuves formelles, des spécifications algorithmiques, des programmes, des preuves de correction de ces programmes, etc. Un nouvel algorithme de vérification d'égalité de types, qui procède autant que possible de manière intentionnelle (c'est-à-dire sans expliciter inutilement la définition des constantes), a permis des performances tout à fait satisfaisantes pour le développement de preuves interactives. Ceci nous a permis d'abandonner le développement de la version C du Calcul, qui ne donne qu'un gain d'efficacité marginal, amplement compensé par les possibilités supérieures qu'offre CAML en tant que métalangage interactif.

### *2.2 SYNTHÈSE DE PREUVES*

Dans le cadre du Calcul des Constructions, Thierry Coquand a travaillé sur l'implantation d'une méthode de preuve automatique de lemmes. Cette méthode s'inspire essentiellement de la méthode des tactiques, du système de preuves de programmes LCF développé aux universités d'Edimbourg et de Cambridge.

### *2.3 EXTRACTION DE PROGRAMMES*

On peut utiliser le Calcul des Constructions pour le développement de programmes. En effet, ce formalisme permet le codage de preuves intuitionnistes d'ordre supérieur et la preuve intuitionniste qu'une spécification est un $\lambda$-terme que l'on peut voir comme un programme. Ce programme termine toujours et contient de manière interne sa preuve de correction.

On aimerait parfois écrire des programmes qui ne terminent pas à priori, mais dont on prouve par ailleurs la terminaison. Christine Paulin a proposé une modification du Calcul des Constructions pour résoudre ce type de problèmes. On distingue explicitement le langage de programmation, le langage de preuve et le langage de développement de programme. On prouve alors (grâce à la définition d'une notion de réalisabilité pour ce calcul) qu'une preuve d'une spécification (i.e. un terme représentant le développement d'un algorithme) permet d'obtenir un programme ainsi qu'une description des propriétés satisfaites par ce programme.
Un prototype de ce système a été réalisé en CAML à partir du système usuel des Constructions et permet l'étude pratique d'exemples.

## Actions industrielles

Notre langage CAML est diffusé par la société ILOG.
Nous sommes en négociation avec le Centre de Recherches Bull en vue d'une convention de collaboration sur le thème "Programmation déclarative".
La Société Hewlett-Packard a mis à notre disposition des postes de travail dans le cadre d'un premier contact en vue d'une collaboration.
F. Fages est actuellement en détachement au Laboratoire de Recherches Thomson-CSF.
A. Suarez a quitté le projet au 1er octobre 87 pour rejoindre le Centre de Recherches PRL de DEC.

## Actions européennes

Nous participons à une action du programme "Stimulation".
Notre projet est engagé dans trois propositions de projets pour la deuxième phase d'ESPRIT dont deux au titre du programme "Basic Research".

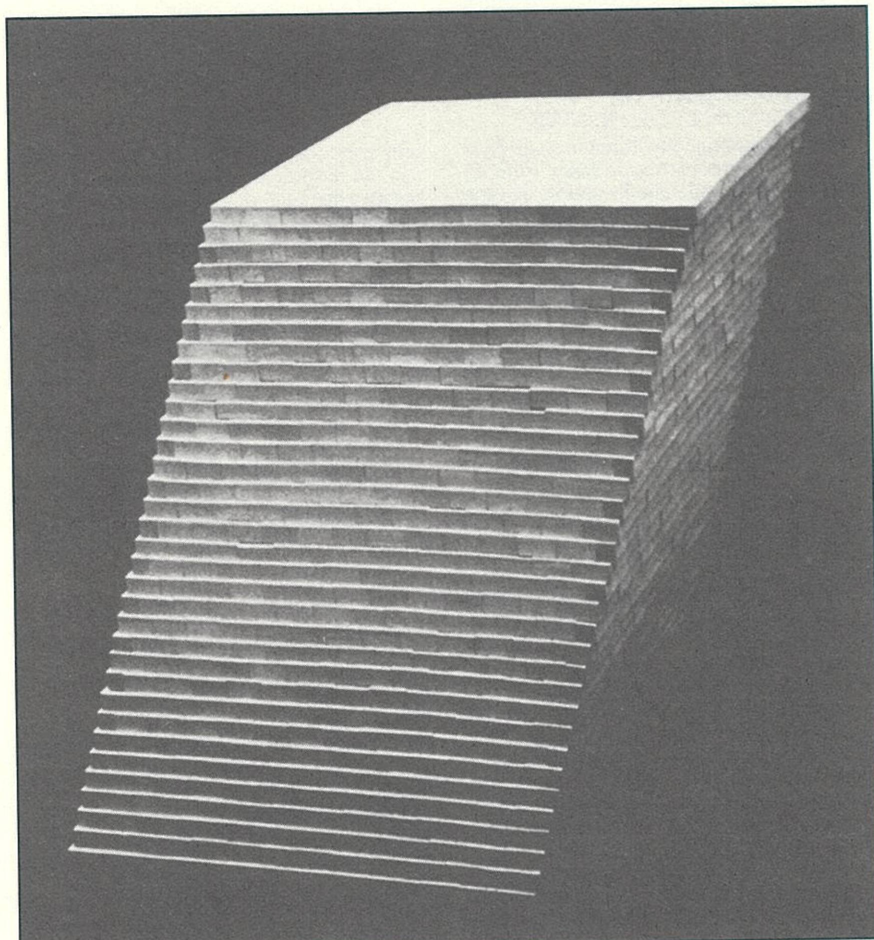
## Diffusion des résultats

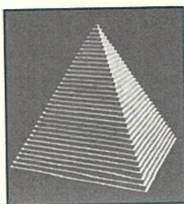
Les résultats obtenus par l'équipe sont largement diffusés dans des colloques nationaux et internationaux et dans des publications. CAML est maintenant distribué par la société ILOG.
G. Huet a organisé un "Institute on Logical Foundations of Functional Programming" à l'Université du Texas à Austin en juin 1987, dans le cadre de la "Year of Programming" de l'Université de Texas. Cet Institut a consisté en un cours de deux jours, enseigné par G. Huet et G. Cousineau, et un séminaire de recherche de trois jours, auquel on participé de nombreux spécialistes du domaine. Les actes en seront publiés chez Addison-Wesley.

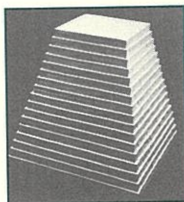


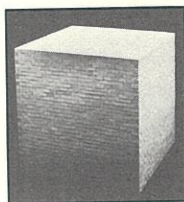


# DE CONCEPTION D'ENVIRONNEMENTS PROGRAMMATION



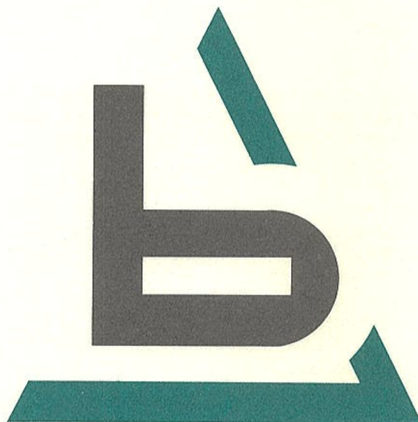






SOL LEWITT
*MAQUETTES POUR SCULPTURE, 1984; RÉALISATION : BRIQUES PEINTES EN BLANC OU COULEUR DE LA BRIQUE LOCALE, HAUTEUR ENVIRON 500 cm. COURTESY GALERIE DANIEL TEMPLON, PARIS.*

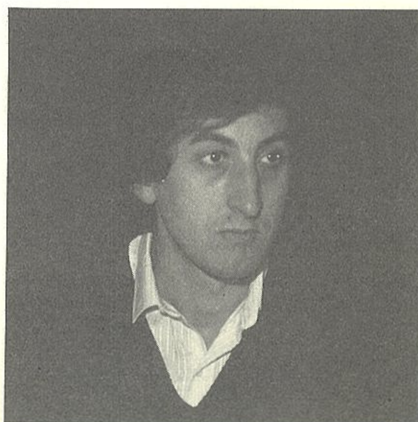
Michel Bidoit



## Génie logiciel : "Environnements de Programmation"

Le Pôle "Environnements de Programmation" regroupe les principales équipes de recherche françaises dans le domaine du Génie Logiciel. Il s'agit là d'un domaine de recherche très actif, dont les applications industrielles sont à la fois nombreuses et importantes. Il est vrai que dans ce domaine les enjeux — y compris économiques — sont de taille, puisqu'il s'agit de maîtriser la production de logiciels, dont plus personne ne méconnaît la part croissante dans tout système informatique. Il n'est donc pas étonnant de constater que les liens entre les équipes participant à ce pôle et le tissu industriel national et européen sont très nombreux. Ces collaborations entre chercheurs et industriels ont souvent lieu dans le cadre de projets ESPRIT (COMETT, GIPE, FORMETOO, METEOR, REPLAY, TOOLUSE), mais sont aussi concrétisées par des contrats de recherche et des boursiers communs recherche/industrie.

La maîtrise de la production de logiciels ne peut s'envisager qu'au travers d'une nécessaire formalisation de cette production. Cette réalité, affirmée depuis longtemps par les chercheurs, est aujourd'hui de plus en plus reconnue par les industriels eux-mêmes. Il est donc indispensable, même dans un domaine en apparence aussi concret que celui de la production de logiciels, de poursuivre des recherches fondamentales sur les différents formalismes qui permettront de mieux maîtriser la production de logiciels complexes, et ceci durant toutes les étapes du cycle de vie du logiciel. A cet égard la qualité des travaux réalisés par les équipes participant au Pôle "Environnement de Programmation" est attestée par de nombreuses publications dans des revues et des conférences internationales de premier plan. Mais ces recherches de nature fondamentale doivent aussi être accompagnées de développements de prototypes qui permettent de valider les résultats obtenus, et constituent également une phase préparatoire au transfert industriel. Dans ce domaine la production des équipes du Pôle "Environnement de Programmation" est très importante et la plupart des systèmes prototypes réalisés font l'objet de valorisations ou sont diffusés dans les principaux centres de recherche français et étrangers.



L'étude des langages, méthodes et outils de spécification formelle constitue un thème central de recherche pour la plupart des équipes impliquées dans le Pôle "Environnement de Programmation". Ce thème est étudié sous différents aspects (les types abstraits algébriques constituant le facteur unificateur), et parmi les résultats obtenus il faut citer les langages de spécification LPG et PLUSS, les environnements de spécification ASSPEGIQUE, OASIS, SACSO. Ces travaux sont accompagnés de recherches sur les méthodes de spécification (projets ASSPRO et MAYDAY) et sur l'impact des spécifications formelles sur les différentes étapes du cycle de vie du logiciel, comme la conception de programmes (projets CPAO, LPG et MAYDAY), le prototypage (projets ASSPRO et LPG), le test de logiciels (projet ASSPRO), la réutilisation de logiciels (projets ASSPRO et CPAO). Un second thème de recherche est la manipulation symbolique de programmes : l'objectif poursuivi est de savoir dériver un environnement de programmation interactif à partir d'une description de la sémantique du langage de programmation et permettre ainsi le maintien de la cohérence des programmes au travers des transformations effectuées (projet CROAP, système CENTAUR). Enfin, un dernier thème de recherche, plus récent mais très prometteur, porte sur l'étude et la réalisation d'outils pour la génération d'interfaces graphiques (projets ASSPRO et CROAP).

ASSPRO *MOTS CLES*

ASSPEGIQUE (environnement/
atelier de génie logiciel)
bases de données pour génie logiciel
Cigale (ASSPEGIQUE)
environnements de programmation
génération d'interfaces
génie logiciel
GRAFFITI (générateur d'interfaces)
interface homme-machine
langage de spécification
METAGRAF (outils de manipulation de réseaux)
méthodes de spécification
PLUSS (langage de spécification)
preuve de spécifications
programmation assistée
prototypage
réseaux de Petri
réutilisation des programmes
Reve (environnement de déduction
automatique)
SADT (méthode de spécification)
spécifications algébriques
spécifications formelles
spécifications par types abstraits
stratégies de test
tests d'intégration
WISH (shell iconique pour Unix)




## Références

**Thèses**

Karsenty S., *"Graffiti : un outil interactif et graphique pour la construction d'interfaces homme-machine adaptables"*, thèse de 3e Cycle, Université de Paris-Sud, Orsay, décembre 1987.

Capy F., *"ASSPEGIQUE : un environnement d'exceptions. Une sémantique opérationnelle des E, R-algèbres, formalisme prenant en compte les erreurs. Un environnement intégré de spécification algébrique : ASSPEGIQUE"*, Thèse de 3e Cycle, Université Paris-Sud, Orsay, décembre 1987.

Kaplan S., *"Spécification algébrique de types de données à accès concurrent"*, Thèse d'État, Université Paris-Sud, Orsay, novembre 1987.

Choquer M.-A., *"Preuve de formules conditionnelles dans les spécifications algébriques conditionnelles"*, Thèse de 3e Cycle, Université Paris-Sud, Orsay, octobre 1986.

Bernot G., *"Une sémantique algébrique pour une spécification différenciée des exceptions et des erreurs : application à l'implémentation et aux primitives de structuration des spécifications formelles"*, Thèse de 3e Cycle, Université Paris-Sud, Orsay, février 1986.

**Livres et revues**

Kaplan S., *"Simplifying conditional term rewriting systems: Unification and Confluence"*, à paraître dans Journal of Symbolic Computation (1987).


Le projet **ASSPRO** (ASSistance à la PROgrammation) a pour objectifs essentiels l'étude des méthodes et des moyens qui permettent de mieux appréhender ce qu'il est convenu d'appeler la "première phase du cycle de vie du logiciel", c'est-à-dire les étapes de conception, de réalisation et de validation du logiciel. Les méthodes liées aux spécifications formelles sont systématiquement utilisées au cours de ces différentes étapes. En particulier, les techniques relevant de l'approche algébrique, ou si l'on préfère des types abstraits algébriques, sont étudiées et mises en application.

De façon plus précise, on peut décomposer les travaux menés dans le cadre du projet **ASSPRO** selon trois axes qui sont :
- l'étude des langages, méthodes et outils de spécification formelle,
- l'impact des spécifications formelles sur différentes étapes du cycle de vie du logiciel, notamment les techniques de prototypage, la réutilisation du logiciel, le test de logiciel,
- l'étude et la réalisation d'outils graphiques pour le Génie Logiciel.

Ces travaux conduisent à la réalisation de prototypes et sont en partie menés en collaboration d'une part avec d'autres équipes de recherche françaises et étrangères (projets EURECA & ALGO, Université de Passau en R.F.A., Equipe de J. Guttag au M.I.T.), d'autre part avec des industriels (C.G.E., Matra, S.F.G.L., C.N.E.S., Hewlett-Packard, Sema-Metra), notamment dans le cadre du projet ESPRIT METEOR.

## Le langage de spécification PLUSS et son environnement ASSPEGIQUE

Les travaux relatifs aux langages, méthodes et outils de spécification formelle associent étroitement des considérations d'ordre pratique et des considérations d'ordre théorique. En effet, la spécification formelle de logiciels de grande taille, devant satisfaire certaines contraintes de robustesse ou de fiabilité (par exemple, les logiciels téléphoniques, les systèmes embarqués ou un système d'exploitation comme Unix), nécessite certaines extensions de la théorie des types abstraits algébriques classiques, lesquelles posent des problèmes sémantiques complexes.

Ainsi, la spécification d'un système de taille importante ne peut être considérée comme un tout, mais doit au contraire être structurée en unités plus petites. De plus, une meilleure structuration facilite la réutilisation de spécifications existantes. Il faut donc disposer d'outils adéquats pour **structurer et modulariser les spécifications formelles.** L'objet du langage de spécification que nous développons, **PLUSS** (Proposition pour un Langage Utilisant des Spécifications Structurées), est précisément d'offrir un certain nombre de primitives (enrichissement, paramétrisation, renommage, contrôle de la visibilité) permettant de structurer et de composer des spécifications algébriques, qui peuvent être **génériques.** Par ailleurs, la spécification d'un système de taille importante est un processus complexe, et il est important de pouvoir prendre en compte au niveau du langage de spécification les différentes étapes du développement de la spécification. C'est pourquoi le langage **PLUSS** permet de distinguer entre des spécifications achevées (les *specs*), dont la signification ne doit plus être remise en cause (leur implémentation étant éventuellement en cours), et des spécifications en cours de conception, encore incomplètes (les *sketchs* et les *drafts*). Le langage **PLUSS** permet donc à la fois d'exprimer des spécifications modulaires et le développement de ces spécifications (grâce à des primitives permettant de convertir des spécifications en cours de conception en spécifications achevées). Enfin, il est important de noter qu'un gros effort a été réalisé pour améliorer la **lisibilité** des spécifications formelles écrites en **PLUSS.**

Les langages et les méthodes de spécification formelle ne sont utilisables que s'ils sont accompagnés d'un ensemble d'outils facilitant leur mise en œuvre. C'est dans ce but que nous développons l'environnement de spécification **ASSPEGIQUE** (ASsistance à la SPEcification alGébrIQUE), qui est un véritable "laboratoire de spécification" intégrant divers outils d'aide à l'écriture, l'analyse et l'utilisation des spécifications algébriques. Dans la conception de l'environnement **ASSPEGIQUE**, un soin particulier a été apporté, d'une part à l'aspect modulaire des spécifications et à la réutilisation des spécifications, d'autre part à la facilité d'emploi, à la flexibilité de l'environnement et à l'interface avec l'utilisateur. Les outils disponibles dans l'environnement de spécification **ASSPEGIQUE** comprennent un éditeur dirigé par la syntaxe, un outil d'intégration de modules de spécification, des outils d'évaluation symbolique, des outils de preuve, un outil d'assistance à l'implémentation de spécifications en packages Ada, et des passerelles vers les logiciels **Reve** et **Slog.** Tous ces outils sont accessibles via une interface plein-écran, multi-fenêtres et accèdent à une bibliothèque de spécifications par un outil spécifique chargé du maintien de la cohérence de cette bibliothèque. Un composant essentiel de l'environnement est le logiciel **Cigale**, un système pour la construction incrémentale de grammaires et l'analyse d'expressions : **Cigale** a été spécialement conçu pour per-

ORSAY

***Responsables scientifiques***
Michel Bidoit et Christian Gresse

***Organisme***
L.R.I. Université d'Orsay
université de Paris Sud
91405 Orsay
☎ (1) 69 41 66 29 & 69 41 66 28

***Participants***
Michel Beaudouin-Lafon
Pierre Bernas
Gilles Bernot
Michel Bidoit
Francis Capy
Christine Choppy
Marie-Claude Gaudel
Christian Gresse
Stéphane Kaplan
Solange Karsenty
Bruno Marre
Thierry Moineau
Françoise Schlienger
Frédéric Voisin
Christina Zoltan

mettre de définir les opérations avec une syntaxe flexible et pour prendre en compte la surcharge d'opérations et la conversion implicite de types, facilités qui concourrent grandement à la lisibilité des spécifications.

Sur un plan plus théorique, les travaux que nous menons sur les types abstraits algébriques portent principalement sur la définition de la sémantique du langage **PLUSS**, sur les extensions nécessaires pour augmenter le pouvoir d'expression des spécifications algébriques (prise en compte des erreurs, des valeurs indéfinies et des traitements d'exception, possibilité de spécifier des accès concurrents à des données partagées), et sur les aspects opérationnels des spécifications algébriques (théorie de la réécriture conditionnelle, compilation de systèmes de réécriture, complexité algorithmique des systèmes de réécriture).

Les perspectives de recherche pour cet axe concernent essentiellement la poursuite des travaux sur le langage **PLUSS**, notamment au travers de diverses études de cas, et la réalisation d'une nouvelle version de l'environnement **ASSPEGIQUE** permettant de prendre en compte les récents développements du langage de spécification.

Par ailleurs, de façon complémentaire à l'évolution de **PLUSS** vers un langage de développement de spécifications, des études sont menées pour **la conception de bases de données pour le Génie Logiciel**. Ces études sont basées sur le modèle *dérivation-verrouillage* et dans un premier temps, une base de données mémorisant les informations concernant le développement de spécifications est à l'étude, pour une intégration au sein de l'environnement **ASSPEGIQUE**.

## La réutilisation de logiciels

La réutilisation de logiciels est pratiquée actuellement sur des bases empiriques, et dans des domaines bien précis (par exemple, des bibliothèques d'algorithmes de calculs numériques). Depuis longtemps, on sait que la clé de toute approche générale à ce problème est l'utilisation de spécifications formelles du logiciel à réaliser et des composants logiciels éventuellement réutilisables.

C'est sous cet angle que nous étudions dans quelle mesure les spécifications algébriques peuvent être utilisées pour la réutilisation du logiciel. Nous avons déjà obtenu des résultats prometteurs, par exemple le résultat suivant :

*Soit une spécification SP' d'un programme à réaliser et soit une spécification SP d'un programme existant. Le programme correspondant à la spécification SP est réutilisable pour la spécification SP' s'il existe une extension SP1 de SP qui est équivalente, à des renommages près, à un enrichissement de SP'* (intuitivement, on ajoute certaines fonctions à SP et on en oublie d'autres).
Les notions de renommage et d'enrichissement étaient déjà connues. L'innovation tient à la notion d'extension et à la sémantique mathématique qui en est donnée : cela permet de définir une modularité des spécifications compatible avec celle des programmes. On obtient ainsi un critère théorique indiquant si l'on peut composer le programme associé à SP et le programme correspondant à l'extension sans que les choix de programmation interfèrent. Cette formalisation permet de plus de définir précisément des notions de réutilisation efficace et de réutilisation non efficace.

Les perspectives dans ce domaine sont évidemment très prometteuses. Dans un premier temps, il faut étudier comment cette formalisation de la notion de réutilisation peut être étendue pour prendre en compte toutes les primitives de structuration des spécifiques algébriques, et notamment la paramétrisation. A terme, ces recherches doivent déboucher sur la définition de bases de composants logiciels réutilisables.

## Le test de logiciel

Une théorie du test de logiciels basée sur des spécifications formelles a été proposée en 1982 par Luc Bougé et appliquée aux spécifications algébriques. Ces travaux ont été poursuivis depuis, d'abord aux Laboratoires de Marcoussis, puis dans le cadre du projet **ASSPRO**, et reposent sur les idées suivantes :
- Les axiomes d'une spécification algébrique étant constitués d'égalités (éventuellement conditionnelles) entre deux termes, on peut soumettre au programme sous test les parties droite et gauche de ces égalités (pour des instances "bien choisies", cf. ci-dessous). Le succès ou l'échec d'un tel test est alors déterminé en vérifiant si les deux résultats obtenus sont égaux. La spécification joue donc un rôle d'oracle, puisqu'elle permet de décider du résultat des tests.
- La structure de la spécification peut être utilisée pour guider le choix des valeurs de test. Par exemple, en introduisant une mesure de complexité sur les types définis, on peut ne tester que des valeurs de complexité inférieure à une certaine borne. On peut aussi donner des valeurs aléatoires aux variables de types prédéfinis (déjà testés). On dit que l'on fait respectivement des hypothèses de régularité sur les types définis et d'uniformité sur les types prédéfinis. Ces hypothèses permettent de sélectionner de façon pertinente des jeux de tests finis et de taille raisonnable.

Kaplan S., *"Rewriting with a non deterministic choice operator"*, à paraître dans Theoretical Computer Science (1987).

Bidoit M., Capy F., Choppy C., Choquet N., Gresse C., Kaplan S., Schlienger F., Voisin F., *"ASSPRO : un environnement de programmation interactif et intégré"*, Techniques et Sciences Informatiques, vol. 6, n° 1, janvier 1987, pp. 21-40.

Gaudel M.-C., Soria M., Froidevaux C., *"Types de données et algorithmes"*, Collection Didactique, INRIA Ed., 1986.

Bougé L., Choquet N., Fribourg L., Gaudel M.-C., *"Test sets generation from algebraic specification using logic programming"*, Journal of Systems and Software, vol. 6, n° 4, novembre 1986, pp. 343-360.

Gaudel M.-C., *"Spécifications formelles et validation de logiciel"*, Revue Enjeux de l'AFNOR, novembre 1986.

Gaudel M.-C., *"Récit de voyage aux pays des tests et des caribous"*, Revue Génie Logiciel, novembre 1986.

Gaudel M.-C., Papon E., *"Outils de développement de prototypes à partir de spécifications algébriques"*, Revue Génie Logiciel, novembre 1986.

Choppy C., *"Techniques et aspects du prototypage"*, Revue Génie Logiciel, n° 3, janvier 1986, pp. 4-12.

Bernot G., Bidoit M., Choppy C., *"Abstract data types with exception handling: an initial approach based on a distinction between exceptions and errors"*, Theoretical Computer Science, vol. 46, n° 1, janvier 1986, pp. 13-46.

Voisin F., *"Cigale: a tool for interactive grammar construction and expression parsing"*, Science of Computer Programming, vol. 7, n° 1, 1986, pp. 61-86.

Anna Gram (ouvrage collectif), *"Raisonner pour programmer"*, Dunod Ed., 1986.

**Colloques**

Beaudoin-Lafon M., Karsenty S., *"Iconic Shells for Multitasking Workstations"*, Proc. ACM Symposium on Personal and Small Computers, Cannes (France), mai 1988 (à paraître).

Gaudel M.-C., Moineau Th., *"A Theory of software reusability"*, Proc. ESOP'88, Nancy, mars 1988.

Gaudel M.-C., *"Spécifications Formelles de Logiciel"*, Journées Francophones sur l'Informatique, Genève, janvier 1988.

Beaudoin-Lafon M., Karsenty S., *"Prototyping Interfaces for Applications Depicted by Graphs"*, Proc. 21st Annual Hansi International Conference on Systems Sciences, vol. 2, pp. 436-445, Koua Hansi (USA), janvier 1988.

Beaudoin-Lafon M., Karsenty S., *"A Framework for Man Machine Interfaces Design"*, Proc. EUUG Autumn Conference, Dublin (Ireland), pp. 1-10, septembre 1987.

Récemment, des notions de couverture de test pour des spécifications algébriques ont été définies. Ces critères de couverture induisent des stratégies de test où les jeux de tests successifs tendent vers une limite qui est une preuve de la correction du programme.

Ces stratégies de test peuvent être exprimées formellement et automatisées partiellement au moyen de démonstrateurs de théorèmes. Cette automatisation est bien évidemment l'objectif premier de nos recherches sur le test de logiciels. Pour l'instant nos expériences sont faites en utilisant des systèmes de programmation logique existants, et diverses stratégies ont été implémentées en **Slog** et en **Quintus Prolog**. Un exemple non trivial, les arbres binaires de recherche, a été traité complètement selon ces stratégies. Une évaluation de la qualité des tests obtenus par injection de fautes dans les programmes est en cours.

Les recherches actuelles portent sur deux points :

- La définition de nouvelles stratégies, en particulier pour les types bornés. Ces types sont traditionnellement testés de manière spécifique (tests systématiques des limites) et on retrouve cette spécificité au niveau théorique : les stratégies classiques s'appliquent mal (hypothèses peu réalistes).
- La réalisation d'un environnement de programmation logique adapté à la production de jeux de test à partir de spécifications algébriques.

Ces travaux devraient conduire à une collaboration avec EDF.

## Le prototypage et le test d'intégration

Lors de la conception de systèmes complexes, chaque module doit être testé individuellement, ainsi que l'intégration des modules connectés entre eux. Les techniques de **prototypage** peuvent considérablement aider les **tests d'intégration.** Partant d'une spécification formelle hiérarchique et modulaire, les parties exécutables de celle-ci peuvent être utilisées comme un prototype du système. On peut ensuite progressivement remplacer les modules de spécifications par leur implémentation. L'exécution d'un tel système, où l'intégration est partielle, nécessite des **techniques d'évaluation mixte** combinant la réécriture (pour les modules uniquement spécifiés) et l'évaluation directe (pour les modules implémentés). Les problèmes liés à l'étude de telles techniques d'évaluation mixte sont en cours d'évaluation.

## L'étude et la réalisation d'outils graphiques pour le génie logiciel

La banalisation des stations de travail munies de possibilités graphiques, de moyens de désignation et de systèmes de multi-fenêtrage permet de mieux prendre en compte les problèmes d'interface utilisateur. Nos travaux portent sur la description et le développement d'interfaces graphiques adaptables et facilement réutilisables, et s'appuient sur nos réalisations précédentes, PETRIPOTE (un outil pour l'édition et la simulation graphique de réseaux de Pétri, qui a été utilisé au L.R.I. et largement diffusé dans différents laboratoires de recherche européens) et le terminal STEP (développé en collaboration avec les laboratoires de Marcoussis sous contrat CNET, et actuellement commercialisé par la société AMAIA).

Nous étudions essentiellement les interfaces graphiques mettant en œuvre des objets structurés, comme par exemple des réseaux de Pétri, des schémas de programme, des types de données, des diagrammes SADT, etc. Dans chaque cas, l'interaction avec l'utilisateur doit être faite directement au niveau de la représentation graphique externe des objets manipulés, même si l'interface graphique doit parallèlement mettre à jour la structure interne correspondante. Les problèmes de performance sont évidemment cruciaux en ce domaine.

Nous avons développé un méta-modèle d'interaction graphique, **UFO** (User-Friendly Objects), centré sur la notion d'objet, permettant de développer rapidement des interfaces graphiques sophistiquées. Au-dessus d'**UFO** sont développés deux types d'outils :

- **METAGRAF**, qui permet de définir la représentation de réseaux complexes et leur manipulation,
- **GRAFFITI**, qui permet la définition graphique interactive de la structure de contrôle de l'interface (menus, boîtes de dialogues, etc.) et la définition interactive des liens entre l'interface et l'application à l'aide de points d'entrée et de valeurs actives.

**GRAFFITI** est l'un des tout premiers vrais générateurs d'interface qui soit à la fois réellement interactif et qui permette la construction d'interfaces adaptables par l'utilisateur. Contrairement à la génération de compilateurs à partir de la description de la syntaxe et de la sémantique d'un langage, l'aspect interactif dans la génération d'interfaces est crucial car il permet un prototypage rapide et une construction facile de l'interface.

**UFO** et **GRAFFITI** sont utilisés pour réaliser **WISH**, un shell iconique pour Unix. **WISH** gère la représentation d'entités Unix telles

que fichiers, directories, processus, utilisateurs, machines, etc. sous forme d'icônes, en associant aux actions de l'utilisateur (déplacements, ouverture, etc.) une sémantique dépendant du type de l'icône mise en jeu. **WISH** permet de réaliser de manière simple et intuitive la plupart des opérations effectuées habituellement à l'aide du shell Unix standard, et il est paramétrable par l'utilisateur.

Les objectifs à moyen et long terme sont l'utilisation des outils mentionnés ci-dessus dans le cadre d'applications de plus grande taille, comme l'environnement de spécification **ASSPEGIQUE**, un environnement de programmation parallèle, etc. Ceci nécessite la définition de nouvelles abstractions, l'extension du modèle de communication entre l'interface et l'application et l'intégration des différents outils au sein d'un atelier de construction d'interfaces.


Bernot G., *"Good Functors... are those preserving philosophy"*, Proc. Summer Conference on Category Theory and Computer Science, Springer-Verlag, LNCS n° 283, pp. 182-195, septembre 1987.

Choppy C., *"Formal specification, prototyping and integration tests"*, Proc. of the 1st European Software Engineering Conference, Strasbourg, septembre 1987, pp. 185-192.

Choppy C., Kaplan S., Soria M., *"Algorithmic complexity of term rewriting systems"*, Proc. of the 2nd International Conference on Rewriting Techniques and Applications, Bordeaux, LNCS n° 256, Springer-Verlag, mai 1987, pp. 256-273.

Kaplan S., *"A Compiler for conditional term rewriting systems"*, Proc. of the 2nd International Conference on Rewriting Techniques and Applications, Bordeaux, LNCS n° 256, Springer-Verlag, mai 1987, pp. 25-41.

Bidoit M., Gaudel M.-C., Guiho G., *"Suggestion pour une programmation systématique et sûre des cas d'exception en ADA"*.

Proc. 9e Séminaire Tuniso-Français d'Informatique "Sûreté de fonctionnement des systèmes informatiques", Tunis (Tunisie), avril 1987.

Gresse C., Raffinat P., *"PROGRAIS: An experimental programming environment for supporting transformational development of software"*, Proc. 1st Conference on Software Engineering Environment, University of Keele (Grande-Bretagne), Ellis Hotwood Ed., avril 1987.

Kaplan S., Rémy J.-L., *"Completion of conditional rewriting systems"*, Proc. of the M.C.C. Workshop on Resolution of Equations in Algebraic Structures, Austin (1987).

Kaplan S., Pnueli A., *"Specification and implementation of concurrently accessed data structures : an abstract type approach"*, Proc. Conf. STACS 87, Passau (RFA), mars 1987, Springer-Verlag LNCS 247.

Doucet A., Gaudel M.-C., *"Bases de données et génie logiciel : vers l'intégration des outils de développement de logiciel"*, Journées AFCET "Bases de Données", La Rochelle, octobre 1986, Eyrolles 1987.

Karsenty S., *"Object oriented tools for the design of high level interfaces: the Key for Adaptability"*, Proc. IFIP WG 8.4 Working Conference: Methods and Tools for Office Systems, octobre 1986, ed. by G. Bracchi and D. Tsichritzis, North-Holland, pp. 67-78, 1987.

Gaudel M.-C., *"Logic programming and automatization of software test strategies"*, Conf. Invitée, Proc. International Conference on Artificial Intelligence: methodology, systems and applications (AIMSA), Varna, Bulgarie, septembre 1986.

Choppy C., Gaudel M.-C., *"Impact des spécifications formelles sur le développement de logiciel"*, Recueil des Conférences de la Convention Informatique, Paris, septembre 1986, tome A, pp. 335-339.

Gaudel M.-C., *"Automation of testing in software development"*, Position Paper, Panel on Automation of Software Development, IFIP 86, Dublin (Irlande), septembre 1986.

Bernas P., *"Environnements de Programmation et Propagation d'exception"*, Proc. 3e Congrès AFCET de Génie Logiciel, Versailles, pp. 269-280, mai 1986.

Bidoit M., Gresse C., Losavio F., Schlienger F., *"Automatic programming techniques applied to software development : an approach based on exception handling"*, Proc. 1st International Conference on Application of Artificial Intelligence to Engineering Problems, Southampton (Grande-Bretagne), avril 1986, pp. 165-177.

Bernot G., Bidoit M., Choppy C., *"Algebraic semantics of exception handling"*, Proc. ESOP, Sarrebruck (RFA), mars 1986, Springer-Verlag LNCS 213, pp. 173-186.

Kaplan S., *"Rewriting with a nondeterministic choice operator: from algebra to proofs"*, Proc. ESOP, Sarrebruck (RFA), mars 1986, Springer-Verlag LNCS 213, pp. 351-374.

Bernot G., Bidoit M., Choppy C., *"Abstract implementation and correctness proofs"*, Proc. 3rd STACS, Orsay, janvier 1986, Springer-Verlag LNCS 210, pp. 236-251.

# MAIDAY
# Outils et Méthodes pour Spécifier et Construire

MAIDAY *MOTS CLES*
aide à la conception
aide à la documentation
aide à la maintenance
dérivation des spécifications
EDME (éditeur méthodique pour l'algorithmique)
environnements de programmation
génie logiciel
Maiday (système de spécification et conception)
MENTOR/MENTOL/METAL
méthode de spécification
prédicats types
réutilisation des programmes
Sacso (système pour les spécifications)
schémas de dérivation
synthèse de programmes
validation par tests



L'objectif général du projet MAIDAY est de contribuer à un travail de réflexion sur la spécification de problèmes et la construction de programmes, et de développer des outils dans ce domaine.
Il est notoire que le coût de développement et de maintenance des systèmes informatiques augmente. Des études ont montré que si le coût de maintenance provient en partie des erreurs de programmation, il résulte surtout d'une mauvaise description des besoins exprimés par les demandeurs : c'est-à-dire d'une mauvaise spécification.
Si l'étape de spécification apparaît alors comme importante, voire primordiale, il n'est pas encore réaliste de penser qu'il suffit de spécifier un problème informatique pour que sa construction effective en découle de façon automatique. Notre projet étudie, au moins en partie, les étapes de spécification et de programmation avec les objectifs suivants :

- Mieux comprendre les mécanismes de base de la construction, en concentrant notre attention sur l'enchaînement des choix cruciaux qui conduisent à une spécification et à un programme.
- Concevoir des outils d'aide puissants :
  - à la spécification,
  - à la programmation directe (à partir d'un énoncé informel) ou indirecte (à partir d'une spécification formelle par mise en évidence de techniques de transformation).
- Réaliser des maquettes de ces outils en utilisant, de façon si possible novatrice, des outils logiciels de recherche, leur fournissant ainsi une base d'expérimentation.

Schématiquement, à côté de la définition de langages de spécification et de programmation, il convient de définir des formalismes permettant d'exprimer des *dérivations* de spécifications et de programmes, ainsi que des *schémas de dérivation* (parfois appelés tactiques ou stratégies). Cette approche précise et généralise les diverses études sur la méthodologie de la programmation, trop souvent objet de descriptions approximatives et imprécises. Elle permet de dépasser le dogmatisme des approches descendantes en autorisant de mêler, selon le contexte, différents points de vue : décomposition hiérarchique, recomposition ascendante, approche guidée par les objets... Ce point de vue permet de considérer une méthode comme un enchaînement particulier de décisions de construction, ainsi, un choix entre de tels enchaînements devient possible. Les conséquences de cette approche sont nombreuses :

- Aide à la documentation interne. La suite des décisions de construction est un élément capital d'aide à la compréhension du rôle et de la structure d'un logiciel.
- Aide à la modification. L'adaptation d'un logiciel à une évolution du cahier des charges initial va consister à remettre en cause certaines décisions de construction.
- Aide à la réutilisation. Plutôt que de généraliser a priori un problème pour le faire dépendre de paramètres en vue d'en fournir ultérieurement plusieurs instances, il est économiquement plus raisonnable de ne résoudre que le problème posé et de réutiliser non pas le logiciel associé, mais la dérivation qui l'a fourni.

La mise en œuvre de ces idées nécessite, d'une part, l'utilisation ou l'adaptation de langages permettant d'exprimer spécifications et programmes, d'autre part, la définition de langages de description de dérivations et de raisonnements. Ces définitions et adaptations se concrétisent par la réalisation de maquettes logicielles permettant de valider les hypothèses émises. Fondés sur les idées précédentes, trois axes qui donnent lieu à la réalisation de maquettes, se sont dégagés : construction raisonnée d'une spécification à partir d'un cahier des charges, construction raisonnée d'un algorithme à partir d'un cahier des charges, construction interactive d'un programme fonctionnel à partir d'une spécification exprimée en logique clausale.

## Sacso : assistance à la spécification

Le but de notre projet est l'étude des mécanismes mis en œuvre lors de la construction de spécifications. La spécification d'un système complexe ne s'effectue pas de manière linéaire mais plutôt par une succession d'étapes d'enrichissement, de validation, d'optimisation ou de modification de la spécification en cours de construction. En reprenant les idées développées en introduction, nous distinguons trois niveaux dans l'étape de construction d'une spécification : le produit ou spécification, le processus de construction de spécifications et les stratégies ayant conduit à l'élaboration de la spécification. Notre travail a essentiellement porté sur l'étude des deux premiers niveaux et à leur mise en œuvre dans le logiciel SACSO, système d'aide à la construction et à la validation de spécifications, [fin87].

### 1. CONTEXTE DU TRAVAIL

Une spécification décrit les besoins exprimés par les utilisateurs, en particulier les fonctionnalités attendues du système et l'interaction entre le système informatique et son environnement. Les spécifications considérées sont définies formellement de manière constructive par enrichissements successifs à partir de spécifications de base disponibles dans une bibliothèque. Il est

Wancy

***Responsables scientifiques***
Jean-Pierre Finance
Jacques Guyard

***Organisme***
C.R.I.N. Université de Nancy
BP 239
54506 Vandœuvre-lès-Nancy
☎ 83 91 21 19

***Participants***
F. Alexandre
M. De Silvestri
D. Galmiche
D. Girardet
J.-P. Jacquot
N. Levy
A. Quéré
J. Souquières

important de préciser que le système et ses outils sont prévus pour travailler sur des spécifications incomplètes, c'est-à-dire en cours d'élaboration [sou86]. Le langage utilisé est un langage basé sur les types abstraits algébriques. L'étude d'une sémantique formelle de ce langage est en cours. Une validation des idées et du système a été réalisée sur des exemples réels dans le cadre d'une collaboration industrielle [sil87]. Le système est actuellement disponible sur SUN. Il est écrit en LeLisp-Ceyx et interfac'e avec X-Windows.

**2. LE PROCESSUS DE CONSTRUCTION**

Le processus de construction suivi par le spécifieur fournit une trace rigoureuse de l'activité de développement. Elle est essentielle pour comprendre, vérifier, maintenir et réutiliser une spécification. Nous considérons un processus de construction comme une succession d'étapes, chaque étape correspondant à l'application d'un opérateur particulier à la spécification. Nous distinguons trois sortes d'opérateurs :

• Les opérateurs de structuration utilisés pour favoriser l'extension et la réutilisation de spécifications [lev87]. Ils sont basés sur les mécanismes d'enrichissement et de paramétrisation et sont implantés dans le système SACSO par l'intermédiaire des fonctions de modification, de renommage basées sur la relation "utilise", sur la possibilité de manipuler des définitions incomplètes et de bénéficier des aides et déductions fournies par le système. La bibliothèque prédéfinie de types de base et de constructeurs de types (ou types paramétrés) munis des opérations usuelles favorise la modularité et la réutilisation de spécifications.

• Les opérateurs de validation et d'analyse utilisés pour vérifier la correction et l'adéquation de la spécification aux besoins des utilisateurs et contrôler que les spécifications vérifient certaines propriétés. Les opérateurs d'analyse sont implantés dans SACSO par l'intermédiaire d'opérateurs de contrôle et d'un langage de requêtes permettant, à chaque étape de la construction, d'avoir des renseignements sur les incomplétudes, les erreurs éventuelles détectées par le contrôle de types, ce qui est indéfini, utilisé par une entité donnée, non utilisé, etc.
Nous distinguons deux types d'opérateurs de validation :
– les opérateurs de conversion d'une spécification dans un autre formalisme facilitant la communication avec le client. Nous disposons d'outils permettant d'extraire des informations sur la spécification tels que le résumé, la description informelle. La réalisation d'outils de visualisation de la spécification dans un modèle relationnel et de visualisation graphique de la structure des types sont en cours de réalisation.
– les opérateurs de production d'un prototype exécutable de la spécification. Nous disposons d'un interprète travaillant sur des spécifications incomplètes et d'un outil de prototypage transformant une spécification complète et correcte en un programme Lisp.

• Les opérateurs de restructuration utilisés pour éliminer la redondance, favoriser la modularité ou simplifier des fragments de spécification. Ils sont basés sur la manipulation de la structure des types [dub87]. Nous avons introduit un certain nombre d'opérateurs prédéfinis. Ils sont utilisés par exemple lors de la conversion dans un autre formalisme.

**3. PERSPECTIVES**

La description des stratégies [anna86] doit permettre d'exprimer les raisons du choix de l'enchaînement des différents opérateurs lors de l'élaboration du processus de construction et de fournir des heuristiques d'application de ces opérateurs. Une première approche a abouti à la réalisation du pilote. Cette étude se poursuit avec le souci de réutiliser aussi bien des spécifications que des processus de construction. Une étude plus fondamentale sur la réutilisation est en cours, en particulier avec une recherche de critères de comparaison d'opérations et de critères d'application de transformations.
L'étude de la sémantique nous permet de préciser les bases formelles du langage et de construire un outil de prototypage plus complet incluant, outre l'interprète actuel, un évaluateur symbolique.

## Maiday : assistance à la programmation

Dans le cadre du projet Maiday, notre travail est focalisé sur l'activité de création des programmes. Des nombreux résultats des recherche abordant ce domaine, nous retenons essentiellement trois points :

• l'intérêt des langages applicatifs pour exprimer les programmes. Ils permettent en effet de structurer naturellement les algorithmes, mais surtout de *raisonner* formellement sur les programmes.

• la nécessité de fournir au programmeur des guides *effectifs* pour l'assister dans sa démarche de création. En effet, la complexité du passage d'un problème à un programme est telle que la planification de cette activité est elle-même un travail complexe.

• la puissance et la qualité de la technologie matérielle et logicielle. En effet, grâce aux éditeurs syntaxiques et sémantiques et aux environnements, il devient possible d'automatiser certaines tâches annexes, de contrôler ou de tester les programmes au cours même de leur création.

Notre objectif est donc de construire des systèmes, suffisamment intelligents, d'aide à une programmation raisonnée en intégrant en un outil unique les trois points précédents. Nous sommes convaincus que l'effet synergétique résultant de l'intégration est un facteur clé pour avancer dans notre domaine de recherche.
Nous avons adopté une démarche expérimentale consistant à construire des maquettes logicielles puis à les évaluer. Chaque cycle expérimental vise trois buts : acquérir une expertise profonde sur les différents points, obtenir des critères objectifs de jugement des différentes techniques (conceptuelles et logicielles) vis-à-vis de la programmation et enfin, mettre en évidence les mécanismes fins de l'activité de création de programmes.

**1. RÉSULTATS OBTENUS**

Une première maquette, EDME, a été livrée en 1984. Elle a été évaluée grâce au travail de J.M. Hoc qui l'a utilisée comme support pour une expérience en psychologie de la planification.
EDME est un éditeur méthodique d'aide à la création d'algorithmes. Nous avons utilisé le langage MEDEE comme support d'expression, la méthode déductive comme guide systématique, et MENTOR-METAL comme base d'implantation. Les caractéristiques d'EDME sont :

• une stricte incrémentalité garantissant que l'algorithme est à tout moment correct sur les plans syntaxiques, contextuels et méthodiques. Les différents contrôles sont effectués par l'éditeur et implantés en MENTOL.

• une aide à l'utilisateur sous la forme d'une construction automatique du texte, d'inférences de textes (déclaration de compteurs...) et de calculs (objets à initialiser, travail restant à faire...).

• une ergonomie de l'interaction soignée, en particulier par le biais d'un multi-fenêtrage sophistiqué.

• un langage de commande destiné à réduire la distance entre les opérations intellectuelles de conception de l'algorithme et les opérations d'édition.

L'expérience de J.M. Hoc, destinée à étudier les stratégies de planification des programmeurs, a consisté à analyser les sessions complètes de travail avec EDME d'une quinzaine de professionnels extérieurs au laboratoire.
L'analyse des résultats de cette évaluation a mis en évidence les points positifs suivants. La conception et la réalisation d'EDME sont robustes. Notre hypothèse sur le contenu d'un outil d'aide (ergonomie, contrôles, inférences, guide systématique, langage de commande) a été confirmée, en particulier par les temps d'apprentissage très courts et la

fourniture d'algorithmes corrects par tous les sujets de l'expérience. Un certain nombre de difficultés ont été mises en évidence. Leur importance est essentielle à notre démarche puisqu'elles définissent nos directions de travail actuelles. Elles peuvent être regroupées en trois classes :

- *L'EXPRESSIVITÉ DU LANGAGE.* Le langage MEDEE implanté utilise des types de données et des structures de contrôle de trop bas niveau vis-à-vis de l'activité de conception. L'introduction d'une construction oblige à définir immédiatement des détails très fins alors que le concepteur souhaiterait travailler sur un schéma générique.
- *LA NON-LINÉARITÉ DES DÉVELOPPEMENTS.* Cet aspect se traduit de deux façons : par des remises en cause fréquentes en cours de construction et par des changements de stratégies de travail. EDME s'est révélé être un frein dans ces deux situations. Dans le premier cas parce que l'incrémentalité oblige à *oublier* le travail remis en cause, dans le second parce que la méthode implantée ne comporte qu'une stratégie. La leçon à tirer est importante car elle concerne moins l'outil que l'idée, implicite à toutes les systématiques de la programmation, qu'un programme peut être construit dans une démarche linéaire située dans un cadre unique. Clairement, cette idée doit êre très fortement nuancée.
- *LA RÉUTILISATION.* Le modèle de développement supporté par EDME suppose que le programme est construit ex nihilo, la réutilisation est vue comme l'emploi sans modification d'un fragment logiciel existant. Il est vite apparu que le concepteur travaille souvent à partir d'une solution existante qu'il adapte au problème posé. Dans ce type de situations, EDME n'apporte aucune aide. De nouveau, la difficulté remonte au plan des systématiques car aucune d'elles ne supporte réellement cette forme de développement.

Depuis 1986, notre travail est donc orienté vers l'étude et la résolution de ces difficultés.

**2. AIDE A LA CONCEPTION DIRECTE DE PROGRAMMES**

Ce volet du projet traite du problème de l'expressivité du langage. Nous y avons adopté une démarche pragmatique en introduisant dans le langage MEDEE des constructions de haut niveau telles que les abstractions de types et les itérateurs. Par rapport à EDME, la maquette, en phase de conception, permettra d'étudier une solution à un problème important, issu de nos activités pédagogiques.

Il apparaît que les langages de programmation mis à la disposition des utilisateurs occasionnels ou débutants peuvent être schématiquement répartis en deux classes : les langages généraux (PASCAL, C), et les langages intégrés dits de *quatrième génération* (MAPPER, Dbase III). Nous avons constaté que les débutants opposent fortement ces deux classes. Dès lors, le passage entre ces deux modes d'expression pour concevoir les programmes est délicat, de plus, il souffre d'un total manque de systématique. Selon nous, la discontinuité vient du fait que la notion de *type*, fondement actuel de la programmation, y est approchée de façon différente :

- dans les langages intégrés, on utilise directement le type via ses opérations de manipulation. On peut enchaîner des opérations mais pas créer de nouveaux types.
- dans les langages classiques, les types servent d'une part à modéliser les objets du problème et d'autre part à les implanter en machine.

L'objectif est alors d'obtenir un outil permettant un passage continu entre ces deux formes d'approches. Il offrira donc les possibilités suivantes :

- constructeurs de types de haut niveau comme PILE ou ARBRE, mais aussi RELATION et FEUILLE DE CALCUL.
- modularité et structuration des analyses dans les phases de programmation et construction automatisée des algorithmes lors de l'enchaînement manuel de *calculs.*
- gestion de l'intendance par la prise en charge des aspects syntaxiques et sémantiques.
- documentation gérée et construite automatiquement.

La définition du nouveau langage est maintenant terminée, ainsi que les tests des outils comme CEPAGE ou MENTOR-TYPOL destinés à supporter l'implantation. Nous espérons disposer d'une maquette en 1988. Nous envisageons d'en étudier une mise en œuvre au plan industriel, dans le cadre de notre participation au projet européen COMETT.

grenoble

# Spécifications formelles & programmation générique

***Responsable scientifique***
Didier Bert

***Organismes***
LIFIA (Lab. d'Informatique Fondamentale et Intelligence Artificielle)
INPG
46, avenue Félix-Viallet
38031 Grenoble Cedex
☎ 76 57 46 64

***Participants***
H. Comon
P. Drabik
R. Echahed
P. Jacquet
D. Lugiez
M.-L. Potet
J.-C. Reynaud

LPG *MOTS CLES*
démonstration automatique
environnements de programmation
langage de spécification
machine abstraite/virtuelle
programmation fonctionnelle
programmation logique
réutilisation des programmes
langage de spécification
spécifications formelles
spécifications génériques
synthèse de programmes
synthèse de spécifications
types abstraits algébriques

## Le projet LPG : un langage de spécification modulaire et générique

Le processus de construction de logiciel a été analysé depuis une quinzaine d'années, et ces études ont mis en évidence la nécessité de réaliser un développement en phases successives ou itérées allant de la constitution du cahier des charges et des spécifications fonctionnelles informelles du système à la réalisation effective des programmes, en passant par les étapes d'architecture ou de conception modulaire, de spécifications formelles ou semi-formelles, de prototypage, de tests structurels ou fonctionnels, de codage, d'intégration, etc. Un certain nombre d'outils sont actuellement disponibles sur le marché pour couvrir quelques-unes de ces phases, bien qu'il n'en existe pas encore couvrant véritablement le cycle de vie complet. Les outils pour la spécification dite "semi-formelle" sont assez avancés et apportent déjà beaucoup à la phase de conception et de documentation. Le projet LPG vise à expérimenter, à l'aide d'un langage et d'outils appropriés, la faisabilité et le champ d'application des *spécifications formelles*, que ce soit au niveau de la facilité d'écriture, de la lisibilité, de la puissance d'expression, de la structuration, des outils d'analyse, de validation, de tests, et également dans un deuxième temps, d'apprécier la possibilité de relier les spécifications formelles aux programmes par des processus automatisés et de favoriser la réutilisation des composants logiciels. A la lecture de ces objectifs, et en regard des moyens limités dont dispose un laboratoire universitaire, il est évident que nous ne cherchons pas à réaliser un outil de type industriel, mais plutôt à fournir une maquette dans laquelle la plupart de ces points peuvent être expérimentés, l'expérience venant confirmer ou infirmer l'adéquation du formalisme, du langage et de la maquette aux objectifs visés. Notre travail se situe donc au stade du pré-développement, participant ou étant très en contact avec les recherches théoriques qui peuvent être menées sur ces sujets.

Un langage pour la spécification formelle doit être fondé sur une sémantique rigoureuse. Les choix que nous avons faits reposent essentiellement sur les travaux de Zilles, Guttag, Goguen, Thatcher, Wagner, Wright relatifs à la spécification algébrique des types abstraits, et sur les travaux de Burstall, Goguen, Meseguer, Ehrig, Bergstra, Wirsing pour la structuration des spécifications, la sémantique de la généricité et les bases logiques.

D'une manière plus précise, notre projet repose sur un langage de spécification modulaire et générique, permettant la programmation fonctionnelle et logique, fondé sur la sémantique des clauses de Horn avec égalité. Cette sémantique englobe d'une manière cohérente la sémantique des clauses de Horn comme dans PROLOG, et la sémantique des types abstraits algébriques avec axiomes équationnels et conditionnels. Grâce à cela, l'utilisateur peut décrire son problème en terme de types (abstraits), de fonctions et de relations. On pense ainsi fournir des moyens qui couvrent une large classe d'applications. Ce langage expérimental, appelé LPG, a été implanté au sein d'un "atelier de spécification" qui fournit un ensemble d'outils dont nous parlerons plus loin.

Pour écrire de "grosses" spécifications, les modules de base doivent pouvoir être définis séparément, ou d'une manière hiérarchique (un module utilise des modules déjà définis), ou encore être composés pour obtenir des spécifications complexes. Quelques techniques de composition de modules ont été proposées dans les langages ou projets CLEAR, CAT (Burstall, Goguen), LARCH (Guttag, Horning), ACT ONE (Ehrig, Mahr), ASF (Bergstra, Heering, Klint), ASL (Wirsing) et au GRECO Programmation : SPRAC (CERT Toulouse), PLUSS (M.-C. Gaudel, LRI). Notre approche a étudié tout particulièrement les mécanismes "d'instanciation" d'un module générique (création d'exemplaires). LPG possède un mécanisme de paramétrisation très poussé qui facilite la structuration et la réutilisation. Dans le même ordre d'idées, la preuve de programme générique n'est plus un raisonnement spécifique à un programme, mais elle consiste à démontrer un théorème dont on a donné les hypothèses dans la partie formelle.

## Un environnement pour écrire et manipuler des spécifications

Autour du langage LPG, nous avons réalisé des outils qui permettent de manipuler les spécifications. Ces outils sont intégrés dans un environnement dont une première version a été implantée sur système MULTICS, et dont une deuxième version a été réalisée, en coopération avec l'université catholique de Louvain, sur station SUN, système UNIX.

Ces outils sont :

• UN INTERPRÉTEUR DE SPECIFICATIONS, BASÉ SUR UNE MACHINE ABSTRAITE A PILE : Il permet de tester et d'exécuter des programmes fonctionnels avec une certaine efficacité, car les fonctions de base sont

## Références

D. Bert, P. Drabik, R. Echahed, *"Manuel de référence de LPG, version 1.8"*, RT 17-IMAG-1-LIFIA (1987).

D. Bert, P. Drabik, *"LPG : stucturation des spécifications et validation sémantique automatisée"*, Bigre+Globule 55 (1987).

J.-C. Reynaud, *"Sémantique de LPG"*, RR 651-I-IMAG-56-LIFIA (1987).

H. Comon, P. Lescanne, *"Equational problems and disunification"*, à paraître en JSC.
J.-L. Rémy, H. Comon, *"How to characterize the language of ground normal forms"*, RR. INRIA 676 (1987).

R. Echahed, *"On completeness of narrowing strategies"*, Proc. CAAP'88, Nancy (mars 1988).

P. Jacquet, *"Program synthesis by completion with dependent subtypes"*, Proc. CADE-9, Argonne, USA (mai 1988).

codées directement dans l'interpréteur. De plus, la méthode de compilation des fonctions génériques n'introduit que peu de surcoût (à la fois à la compilation et à l'exécution) par rapport aux fonctions non génériques.

• UN RÉSOLVEUR D'EXPRESSIONS LOGIQUES : Une expression logique est une égalité de termes, ou un prédicat muni de ses arguments. Le résolveur, connaissant les définitions des opérateurs et prédicats, calcule les valeurs des variables qui satisfont une expression logique donnée. La méthode repose sur l'algorithme de résolution de Robinson, et sur la relation de "narrowing". Cet outil permet de faire de la programmation logique en LPG, alors que l'interpréteur est dédié à la programmation fonctionnelle.

• DES ALGORITHMES DE COMPLÉTION DE THÉORIES (ALGORITHME DE KNUTH ET BENDIX) DANS LE CAS ÉQUATIONNEL ET DANS LE CAS INDUCTIF : Comme il a été montré par Musser, Huet, Hullot, cet algorithme permet de démontrer des théorèmes dans des théories équationnelles avec constructeurs, ou satisfaisant à des conditions dites de "complétude inductive". Des améliorations et extensions des techniques de complétion sont intensivement étudiées au CRIN (Nancy).

• UN ÉDITEUR SYNTAXIQUE : Construit à partir du générateur "Cornell Synthesizer" (sur version SUN uniquement).

• UN DÉMONSTRATEUR DE THÉORÈME : pour les spécifications ne rentrant pas dans le cadre des preuves par complétion, il est nécessaire de disposer de démonstrateurs interactifs procédant à partir de règles d'inférence. Sur la version MULTICS, nous avons réalisé une interface avec le système OASIS du CNET qui réalise de telles preuves. Sur la version UNIX, nous sommes en train de concevoir et d'implanter un démonstrateur utilisant la même approche, mais plus facilement modifiable et manipulant les preuves comme des objets.

## Des études théoriques sur les spécifications formelles

Des études théoriques sont poursuivies dans notre équipe sur des sujets liés aux spécifications formelles. On peut les grouper en quatre thèmes : la programmation fonctionnelle avec opérateurs de second ordre, la génération de spécifications à l'aide d'algorithmes de complétion, la résolution d'équations et de "diséquations", les applications de la théorie des institutions.

• LA PROGRAMMATION FONCTIONNELLE : Dans un premier temps, nous avons étudié le langage de programmation fonctionnel FP (de J. Backus) et l'algèbre des programmes dérivée de ce langage. Les résultats de cette approche nous paraissent insuffisants car ils ignorent la notion de type. Les travaux que nous avons entrepris conduisent à une nouvelle conception d'un langage fonctionnel fortement typé. Celui-ci comprend : (1) la définition des types abstraits ; (2) la définition d'opérateurs de second ordre qui génèrent les fonctions "inductives" sur ces types ; (3) les règles d'inférence associées aux opérateurs de second ordre pour prouver des propriétés des programmes ; (4) des règles de transformation donnant des règles d'équivalence des programmes, c'est-à-dire une algèbre des programmes.

• SYNTHÈSE DE SPÉCIFICATIONS ET DE PROGRAMMES . Les activités de notre groupe en synthèse de programme consistent à explorer l'analogie qui existe entre preuves et programmes. L'idée générale est qu'un programme peut être extrait de la preuve de sa spécification exprimée dans une logique adéquate. A cet égard, le langage LPG fournit un environnement qui offre au moins deux possibilités intéressantes :

- appliquer le paradigme "preuve = programme" dans le cadre des spécifications algébriques, l'outil de preuve étant la procédure de complétion de Knuth-Bendix,
- exploiter la notion de généricité, une des caractéristiques fondamentales de LPG, pour poser correctement le problème de la synthèse au "bon niveau" d'abstraction.

Plus précisément les recherches actuelles sont regroupées autour de deux thèmes : (1) l'influence du choix des constructeurs sur la conception et l'efficacité des fonctions écrites sur des types abstraits, et les transformations automatiques entre plusieurs de ces choix ; (2) la synthèse des fonctions spécifiées par clauses de Horn : à partir d'une description d'un opérateur par clauses de Horn, le système produit une spécification éventuellement conditionnelle de cet opérateur.

• ÉQUATIONS ET DISÉQUATIONS : Les travaux portent sur la résolution d'équations (=) et de diséquation (≠) dans les algèbres de termes, les variables étant soit existentiellement, soit universellement quantifiées. Cela contient en particulier la théorie de l'unification et généralise les résultats de Colmerauer pour Prolog 2. Un formalisme et une sémantique des problèmes équationnels ont été décrits (collaboration avec P. Lescanne du CRIN) et il est montré que tout système peut se ramener à un système en forme normale. Un tel système est soit vide, soit est simplifié au maximum (plus de diséquations si possible), ne contient plus de paramètre, et admet toujours au moins une solution. Les résultats sont valables dans la théorie vide, ainsi que dans certaines classes de théories

équationnelles. Les applications sont nombreuses : on peut citer le problème de la complétude suffisante dans le cas des spécifications algébriques et celui de la réductibilité inductive dans le cas de preuves par induction. De même, la transformation de spécifications avec équations entre constructeurs en spécifications "order-sorted" sans équations entre constructeurs est possible. Une autre application est celle de la négation en programmation logique où les diséquations servent à résoudre des problèmes de compléments qui vont autoriser à traiter des buts négatifs quelconques alors qu'on ne savait traiter que des buts négatifs clos (SLDNF résolution).

• THÉORIE DES INSTITUTIONS : La théorie des institutions a été développée par Burstall et Goguen pour rendre compte des relations entre les formalismes logiques et les classes de modèles de ces formalismes. La logique du premier ordre est une institution, de même que la logique équationnelle ou la logique des clauses de Horn. Les liens entre des logiques "comparables" sont décrits à l'aide de "morphismes d'institutions". Nous avons appliqué cette théorie pour comprendre un mécanisme d'évaluation "dynamique" implanté dans un programme de calcul formel. Ce programme est destiné à faire des calculs dans des structures comme des corps, à l'aide de méthodes applicables à des structures plus faibles comme les anneaux. La possibilité de continuer ou non un calcul est automatiquement discutée par le système, et le contexte d'exécution est changé en fonction de cette discussion.

## Actions dans le cadre ESPRIT

Participation au projet FOR-ME-TOO (Project ESPRIT n. 283) : "Formalisms - Methods - Tools: sofware development based on and supporting the concept of reusability of components" en 1987. En collaboration avec la division RTT d'INFIGENIE et en sous-traitance de SYSECA, nous avons défini un ensemble de primitives de structuration de composants algébriques. Cet ensemble a servi à la réalisation de la spécification algébrique d'un gestionnaire de fenêtres pour le logiciel d'un écran haute résolution. Ces primitives peuvent être considérées comme une extension du projet LPG dans le sens d'une programmation par composants. Elles fournissent des opérations de structuration horizontale et verticale.

Participation au projet REPLAY (Project ESPRIT n. 1651 [1598]) : "replay et evaluation of software development plans using higher-order metasystems" et collaboration avec CISI Ingénierie, le CERT-DERI, l'université catholique de Louvain, le CRI (Danemark), la société Alpha (Grèce) et E2S (Belgique). Nous étudions les techniques "bottom-up" d'élaboration et de réutilisation de plans de développement de logiciels, ces plans étant considérés au niveau macroscopique comme des assemblages de composants. Le langage d'expérimentation est LPG, et nous recherchons à maîtriser les effets des modifications d'un composant dans un système complexe où ce composant n'est qu'une partie. Les liens entre les composants peuvent être hiérarchiques (importation), d'instanciation, de représentation, etc. La suite de notre programme de travail est consacrée au développement de programmes algorithmiques à partir de spécifications, et à l'utilisation d'un formalisme de méta-niveau pour décrire les opérations sur les composants.

## Perspectives futures

L'atelier d'analyse de spécifications construit autour de LPG sur stations SUN sera complété par des outils fondamentaux, comme le démonstrateur de théorèmes, et d'autres outils seront améliorés, en particulier le résolveur d'expressions logiques. Le langage LPG sera modifié pour prendre en compte la programmation et la spécification par composants. Il est prévu également des extensions vers la programmation algorithmique (ADA) et développement des programmes.

Sur les parties théoriques, on envisage la poursuite des études sur les systèmes de réécriture et les applications de la résolution d'équations et de diséquations, en collaboration avec les autres équipes du PRC travaillant sur ce thème. Les recherches sur l'analyse des spécifications et la synthèse de fonctions seront approfondies, ainsi que les travaux sur les applications de la théorie des institutions.

Un certain nombre de résultats pourront faire l'objet d'implantations expérimentales dans l'atelier d'analyse de spécifications construit autour de LPG. Ce sont essentiellement la synthèse de fonctions par la méthode de complétion et la programmation fonctionnelle par opérateurs de second ordre.

L'objectif fixé au terme de ce programme de travail est de mettre à la disposition des utilisateurs, d'une part un seul formalisme qui englobe des logiques différentes, et d'autre part une palette d'outils adaptés à ces logiques ou fournissant des possibilités diverses d'expérimentation sur les spécifications (par exemple : évaluation, prototypage, vérifications, génération de programmes, synthèse...).

## Contacts internationaux et industriels

Collaboration avec l'université catholique de Louvain-La-Neuve (Belgique) pour la réalisation de l'atelier de spécification LPG sur SUN. Depuis 1987, cette collaboration est subventionnée par le CNRS et le CGRI.

Contacts avec SYSECA et SIEMENS et réalisation d'une étude de cas de spécification algébrique dans le cadre du projet ESPRIT FOR-ME-TOO : "Formalisms - Methods - Tools: software development based on and supporting the concept of reusability of components" (1987).

Participation au projet ESPRIT REPLAY : "Replay and evaluation of software development plans using higher-order metasystems" et collaboration avec CISI Ingénierie, le CERT-DERI, l'université catholique de Louvain et le CRI (Danemark) de 1988 à 1990.

# Développements d'outils pour la manipulation de spécifications formelles

OASIS *MOTS CLES*
bases de données
interface homme-machine
logique temporelle
preuve de théorèmes
prolog
réécriture
représentation des types
spécifications algébriques
spécifications formelles
spécifications par types abstraits
système de commutation
systèmes distribués
temps réel
types abstraits algébriques
validation de spécifications



## Objectifs scientifiques

La division CLC du Centre Paris A travaille dans le domaine de la commutation et en particulier, elle participe à la définition des cahiers des charges des systèmes de commutation développés par les industriels à partir de ces cahiers des charges.

Une part importante de la réalisation de ces systèmes consiste en développement de logiciels. Le logiciel d'un autocommutateur représente couramment un volume de plusieurs centaines de milliers de lignes de programme.

Les coûts de développement et de maintenance de ces logiciels atteignent de telles proportions qu'il devient urgent de concevoir des chaînes de production de logiciels plus sûres, donc plus automatisées ou plus assistées. Les spécifications sont actuellement écrites en langue naturelle et souffrent de tous les défauts inhérents à ce type de représentation : ambiguïtés, incohérences, omissions.

Il a donc paru nécessaire d'étudier des possibilités de représentation plus formelle de ces spécifications afin de pouvoir effectuer un certain nombre de contrôles de cohérence sur celles-ci, mais aussi de s'appuyer sur ces spécifications pour les étapes ultérieures de la réalisation des logiciels : conception, programmation, mise au point, recette.

L'objectif des études est donc de définir des formalismes de spécification adaptés aux domaines qui nous intéressent, et également les méthodes à mettre en œuvre pour utiliser ces formalismes avec un maximum d'efficacité et les outils permettant d'appliquer ces méthodes avec sécurité et confort. Afin de valider notre approche, il est nécessaire de faire la preuve que l'application de tels formalismes est viable sur des exemples significatifs de notre domaine. Ce domaine recouvre en fait quelques sous-domaines que l'on retrouve dans de nombreux systèmes informatiques : processus "temps réel", gestion de données, dialogue homme-machine.

## Etat actuel

Les études faites ont abouti à la réalisation d'un certain nombre d'outils :

- OASIS : outil de manipulation de types abstraits,
- interprète PROLOG,
- GENTIANE : outil de vérification de formules de logique temporelle,
- OVAL : outil de validation de spécifications de systèmes distribués écrites en langage LDS normalisé par le CCITT.

En parallèle, des résultats théoriques ont été obtenus sur un certain nombre de points, parmi lesquels on peut mentionner :

- relations entre les techniques de réécriture (Knuth-Bendix) et les techniques de résolution,
- relations entre logique temporelle et automates,
- techniques de résolution appliquées à la logique temporelle.

### LE SYSTÈME OASIS

OASIS est un système de manipulation de types abstraits basé sur la notion de spécification algébrique. OASIS permet de traiter deux types d'objets : les types qui sont des spécifications proprement dites et les représentations qui sont des implémentations d'objets à partir d'autres objets.

Initialement développé sur un microprocesseur EXORCISER 6800 de Motorola, dans une version marseillaise de PROLOG, il a été transporté sur MULTICS pour des raisons de performance, ce qui a nécessité la réécriture de l'interprète PROLOG.
Les principales fonctions d'OASIS sont :

- la saisie interactive de types et de représentations,
- la compilation de types et de représentations,
- le contrôle de type,
- l'évaluation symbolique des expressions de types ou de représentations, à l'aide d'un algorithme de réécriture,
- la preuve de théorèmes sur un type ou une représentation, par réécriture, induction, ou traitement par cas,
- la preuve de conformité d'une représentation d'un type par rapport à sa spécification,
- la vérification et la terminaison du système d'équations (algorithme de KNUTH-BENDIX).

Mentionnons en particulier les facilités suivantes :

- interface utilisateur très ergonomique grâce à l'introduction d'un système de multifenêtrage permettant l'utilisation de menus et d'une souris pour les désignations. Cette facilité est disponible sur la version SM90/UNIX d'OASIS.
- prouveur de théorèmes sophistiqué : en particulier, la cohérence des systèmes d'assertions générés lors d'un raisonnement par cas est vérifiée automatiquement, et de plus ces systèmes sont simplifiés au maximum. L'idée de base utilisée est l'application de l'algorithme de Knuth-Bendix aux assertions, considérées comme des équations sans variables.

Enfin, une thèse est en cours sur la construction de programmes à partir de spécifications formelles par types abstraits algébriques. Un langage intermédiaire permettant une des-

***Responsable scientifique***
Gérard Barberye

***Organisme***
CNET - Centre Paris A - Division CLC
(France TELECOM)
38-40, rue du Général-Leclerc
92131 Issy-les-Moulineaux
☎ (1) 45 29 47 65

***Participants***
Ana-Rosa Cavalli
Pierre Combes
Georges Denis
Véronique Lefort
Max Michel
Etienne Paul
Mohamed Abdellaziz

cription abstraite d'algorithme a été défini, et les outils associés aux différentes étapes de passage d'une spécification à un programme sont en cours de définition.

**LE CHOIX DE PROLOG**

Pour réaliser un outil tel qu'OASIS, faisant appel aux techniques de manipulation symbolique et de réécriture, il nous a paru important de prendre un langage adapté. Notre choix s'est alors porté sur PROLOG. La première réalisation d'OASIS avait été faite sur un microprocesseur 6800, à l'aide d'une version de PROLOG développée par le GIA de l'université de Marseille-Luminy, mais des problèmes de performances nous ont conduits à redévelopper un interprète PROLOG en PASCAL (pour des raisons de portabilité) sur un gros calculateur.

Une première version de PROLOG/CNET était disponible en novembre 1981 sur MULTICS, et la dernière version développée par le CNET Paris A date de juin 1985.

Les caractéristiques originales de cette version sont :

- l'introduction de la notion de modularité,
- la possibilité de manipuler diverses formes du programme : texte source, forme objet (postfixée), image mémoire,
- la possibilité de manipuler un programme ou une partie de programme en tant qu'objet PROLOG (terme),
- l'accès au niveau PASCAL à une interface permettant à l'utilisateur de configurer sa version et de créer ses propres prédicats évaluables.

Divers utilitaires ont été développés autour de cette version : éditeur structurel, outil de trace, outil d'analyse statique et également en liaison avec OASIS, une interface bitmap (multifenêtrage, désignation par menus et souris) utilisant le serveur d'affichage APOTRE développé dans le cadre du projet CONCERTO.

L'équipe a également étudié et réalisé une carte coprocesseur PROLOG pour la SM90, qui donne un gain d'environ 4 en temps d'exécution, la version de PROLOG associée restant parfaitement compatible avec la version standard PASCAL.

**UN OUTIL DE SPÉCIFICATION : GENTIANE**

L'outil GENTIANE permet de spécifier des systèmes de processus parallèles, et de prouver des propriétés dynamiques sur ces systèmes, en utilisant comme formalisme la logique temporelle, et comme technique de preuve, l'extension du principe de résolution à cette logique (travaux de Fariñas Del Cerro).

Cet outil réalise les fonctions suivantes :

- saisie interactive de spécifications ; la syntaxe des propriétés temporelles et la saisie de ces propriétés ont été conçues pour être proches de celles d'OASIS,
- introduction d'opérateurs plus évolués permettant une spécification plus lisible,
- mise sous forme normale de chaque propriété avec détection d'une incohérence ou d'une tautologie,
- preuve de propriétés sur la spécification par une technique de résolution, celle-ci pouvant être contrôlée par l'utilisateur.

Cet outil a été étendu de façon à pouvoir manipuler des formules de logique temporelle comprenant des opérateurs portant sur le passé et sur le futur.

**UN OUTIL DE VALIDATION : OVAL**

L'outil OVAL permet d'effectuer un certain nombre de validations sur des spécifications de systèmes de processus parallèles décrites dans le langage LDS.

Il faut signaler que le CNET a lancé des actions importantes en validation de logiciel en collaboration avec des industriels des télécommunications, ce qui peut être l'occasion de transférer le savoir-faire de l'équipe vers le monde industriel.

L'équipe participe également, dans le cadre de la phase principale du programme européen RACE, au projet SPECS concernant les environnements de spécification pour les systèmes de communication et ayant pour échéance fin 1992.

## Réalisations logicielles disponibles

Quatre logiciels sont actuellement disponibles : PROLOG, OASIS, GENTIANE et OVAL.

L'interprète PROLOG/CNET est disponible sur MULTICS et il peut être mis gratuitement à disposition pour les besoins de la recherche et de l'enseignement publics. Il a déjà été distribué à un certain nombre de centres : MULTICS INRIA à Rocquencourt et à Sophia-Antipolis, CICB, CICG, CICT,CICRP, CITI, Ecole des Ponts et Chaussées.

Cet interprète a fait l'objet, en décembre 1983, d'un contrat de licence avec la société CRIL en vue de son industrialisation et de sa commercialisation sous le nom de PROLOG/P. Il est actuellement disponible sur une dizaine de types de machines de puissances diverses. L'équipe collabore avec la société CRIL pour définir les extensions qui sont faites à cet interprète. Il faut de plus noter qu'un compilateur est actuellement en cours de réalisation.

OASIS est actuellement disponible sur MULTICS, sur VAX/VMS, sur SM90/SMX(MPX) et sur SUN. GENTIANE est disponible sur MULTICS. IL peuvent être mis gratuitement à disposition pour les besoins de la recherche et de l'enseignement publics. Il est nécessaire de disposer de PROLOG/P pour pouvoir implanter ces outils. OASIS a été distribué au CILG, aux PTT Suisses, à l'ENST Brest et à l'université de Clermont-Ferrand.

OVAL est actuellement disponible sur SM90/SMX et sur SUN. Il nécessite de disposer de l'infrastructure CONCERTO et de PROLOG/P.

## Contacts hors PRC et contacts internationaux

**Contacts liés à des contrats DGT/DAII ou CNET**

- LCR Corbeville : langage L,
- Laboratoire de Marcoussis : langage PLUSS
- IMAG/LIFIA : langage LPG
- Ecole des Mines de Sophia : Esterel

**Contacts liés à des fournitures d'outils**

- Société CRIL : licence PROLOG/P.

SPRAC *MOTS CLES*
atelier de génie logiciel
bases de données projet
CONCERTO (atelier de génie logiciel)
environnements de programmation
FI (système d'information)
génération de compilateurs
langage de développement
logique des prédicats
méthodes de conception
preuve de programme
preuves de spécifications
représentation des types
réutilisation des programmes
réutilisation des spécifications
schémas conceptuels de systèmes d'information
spécifications algébriques
spécifications par types abstraits
stratégies de preuve
transformation de programmes
transformation des spécifications
types abstraits algébriques

## Références

[1] J. Foisseau et al., *"Program development with or without coding"*, Proc. of IFIP 80, pp. 327-330, 1980

[2] J. Foisseau et al., *"Le système SPRAC : expression et gestion de spécifications, d'algorithmes et de représentations"*, TSI, vol. 4-3, pp. 237-254, 1985

[3] V. Royer, *"Transformations de sémantiques dénotationnelles de langage de programmation en compilation dirigée par la sémantique"*, Thèse de Doctorat de l'UPS, 1986

[4] P. Bernard, *"Synthèse d'algorithmes à partir de spécifications relationnelles : développement de stratégies"*, Thèse de 3e cycle, ENSAE, 1985

[5] R. Jacquart, *"Transformational tools used in a software engineering environment"*, proc. of SE-86 conference, IEE, 1986

[6] H. Horgen, *"TOOLUSE: an advanced support environment for method driven development and evolution of packaged software"*, proc. of the 2nd ESPRIT Technical Week, 1985

[7] J.L. Durieux, *"Ebauche de formalisation de la construction de Jackson-Hughes"*, Congrès CGL3, 1986

[8] D. Dzierzgowski and E. Gregoire, *"Formalising software development methods"*, proc. of the COMPEURO-88 conference, 1988

[9] M. Sintzoff et al., *"Definition 0.1 of the approximation DEVAO of a development language"*, Internal Report, Tool Use, TD.deva01.DD88a, 1988

[10] J. Cazin et al., *"The F1 formalism, an extension of the entity-relationship model using the first order logic"*, Proc. of the 4th I.C. on entity-relationship approach, Chicago, 1985



### UN OBJECTIF : PLUSIEURS PROJETS

L'objectif général de l'équipe est d'étudier et d'assister par des outils informatiques les méthodes formelles de Conception de Programmes Assistée par Ordinateur (1). Cet objectif général a conduit l'équipe à poursuivre ses recherches à travers plusieurs projets, tous dans la même lignée. Chronologiquement, SPRAC de 1979 à 1984, TOOLUSE de 1984 à 1989, et en enfin REPLAY, son projet compagnon débuté en 1986.

## Le projet SPRAC

SPRAC [2] est un Système de Programmation par Réutilisation Assistée de Connaissances. Il permet la spécification formelle de fonctions et de types abstraits dans le but de produire et de valider des composants logiciels. SPRAC est principalement concerné par le développement de programmes classiques écrits en langages impératifs tel PASCAL.

### LES CARACTÉRISTIQUES

SPRAC est basé sur les principes suivants :

- la possibilité de décrire les *objets* du logiciel à trois niveaux distincts, correspondant à des définitions de plus en plus proches de l'implémentation,
- la présence d'une *base de données projet* (BDP) qui se comporte comme un espace de travail permanent ou temporaire. Le modèle de la BDP est relationnel. De ce fait, il supporte une description statique et dynamique des objets et de leurs interactions qui peut être facilement remise en cause pour être adaptée à de nouveaux types d'objets et de développements,
- l'existence d'un ensemble d'outils fournissant une assistance active au développement. Cette assistance s'appuie sur des techniques basées sur l'approche transformationnelle.

Ces trois caractéristiques sont prises en compte au travers du modèle de développement et de son exploitation par les outils correspondants. Plus précisément, le développement de logiciel est organisé d'une manière descendante à l'aide de trois formalismes différents :

- au plus haut niveau, le langage LF, basé sur une logique typée des prédicats du premier ordre sert à spécifier des *fonctions, des types abstraits et des représentations de types*. Tous ces objets sont génériques,
- au niveau intermédiaire, LA est un langage applicatif utilisé pour décrire des *algorithmes,*
- enfin, au troisième niveau, LM considéré comme un langage de programmation dans lequel sont décrits les *modules opérationnels.*

Il a été décidé d'introduire trois niveaux de langages afin d'isoler au mieux les problèmes qui se posent lors du développement de logiciels, et qui sont en général traités simultanément dans un seul et même langage. Brièvement rappelés, ces problèmes sont :

- Quelles sont les *abstractions* utilisées pour résoudre un certain problème ? Avec LF, les représentations de données (les structures concrètes) et le contrôle ne sont pas exprimés, ce qui permet de ne s'intéresser qu'au *QUOI* de la solution cherchée !
- Quel est le *flot de contrôle* ?
LA, langage algorithmique abstrait, introduit les structures classiques des langages applicatifs et donc permet d'exprimer le *COMMENT* de la solution retenue,
- Assumant les deux points précédents, quelles sont les meilleures *structures de données* ?
Les liens entre les types abstraits et les types concrets sont supportés par la BDP elle-même,
- Comment utiliser au mieux la *mémoire* du calculateur ?
C'est au niveau du langage de la programmation que sont résolus les aspects *gestion de la mémoire, partage de la mémoire, etc.*

LF et LA ont été conçus et implémentés pour faciliter les étapes de spécification rapide et de description d'algorithmes. Comme LF et LA sont interprétables, leur utilisation permet d'obtenir très rapidement des maquettes et des prototypes. L'utilisation de LM conduit à la réalisation du produit final.

### L'ENVIRONNEMENT

L'environnement construit intègre les éditeurs syntaxiques associés à LF et LA, les compilateurs correspondants, et principalement le gestionnaire de la Base de Données Projet qui constitue ainsi le noyau de SPRAC. En particulier, la BDP permet la définition et la gestion des liens statiques et dynamiques entre les objets décrits à des niveaux différents. Ces liens sont établis suivant deux dimensions :

- *DIMENSION SPATIALE*
Les objets sont liés par des relations telles que *utilise, réalise, représente, implémente, instancie,...* Tous les objets et leurs liens appartiennent à une entité appelée *VERSION* qui correspond à une vue instantanée du projet.
- *DIMENSION TEMPORELLE*
Dans le but de garder une trace effective du développement d'un logiciel, la généalogie est préservée dans une entité appelée *HISTORIQUE* qui est en fait l'arbre des *VERSIONS.*

### QUELQUES RÉALISATIONS

Parmi les réalisations effectives autour de SPRAC, trois principaux résultats doivent être mentionnés :

Toulouse

***Responsable scientifique***
R. Jacquart

***Organisme***
ONERA-CERT
Département d'Etudes et de Recherches en Informatique
2, avenue E.-Belin
B.P. 4025 - 31055 Toulouse Cedex
☎ 61557111
Fax : 61557172, Télex 521596 ONECERT

***Participants***
J. Cazin
M. Lemoine
P. Michel
V. Royer
J.-L. Durieux
P. Maurice
N. Boulaya
F. Vivares

• *L'ASSISTANCE A LA PRODUCTION DE COMPILATEURS [3]*
Les travaux menés ont abordé ce problème sous l'angle de l'obtention de compilateurs à partir de la définition de la sémantique dénotationnelle des langages de programmation. En d'autres termes, comment obtenir une définition plus concrète (sémantique pour une machine à pile, à registres, etc.) et comment utiliser ce processus de concrétisation comme processus de génération ? La mise en évidence de règles de transformation entre une sémantique source et une sémantique cible permet d'obtenir des compilateurs valides par *construction,*

• *L'ASSISTANCE A LA TRANSFORMATION AUTOMATISÉE DE SPÉCIFICATIONS*
La BDP constitue un modèle du développement de logiciel, dans lequel des états et des transitions entre états ont été formellemnent identifiés. Une plus grande assistance aux utilisateurs est apportée par des outils réalisant certaines des transitions possibles. En particulier, (4) présente la transformation LF vers LA par synthèse automatique, et (5) donne un algorithme sous-optimal du passage **automatique** de LA vers LM.

• *LE SYSTÈME D'INFORMATION F1 [10]*
Le formalisme F1 a été conçu pour permettre la description de schéma conceptuel de systèmes d'informations. C'est aussi un environnement complet qui permet la création et la manipulation de bases de données dans un formalisme uniforme. Ce dernier est basé sur :

- une extension du modèle entité-relation permettant la prise en compte de la hiérarchie des types,
- une logique typée du 1er ordre favorisant l'expression aisée de la recherche d'informations et de contraintes d'intégrité à satisfaire.

F1 et son environnement PENELOPE constituent un moyen efficace de prototypage d'environnements de logiciels, tant sur le plan des concepts qu'il permet d'appréhender que sur le plan des aspects purement machines. A titre de validation, SPRAC a été entièrement réécrit avec F1, avec une efficacité certaine.

## Le projet TOOLUSE

Le projet TOOLUSE est né de la volonté d'inclure dans un environnement tel que SPRAC des langages et des outils de support de méthodes de développement. En effet, le modèle de développement supporté par la BDP de SPRAC étant relationnel, il est possible d'exprimer des contraintes d'intégrité entre les objets d'une version. Cependant, ces dernières ne permettent que l'expression de ce qui doit être satisfait à *posteriori,* et non ce qui doit être satisfait à *priori.* En d'autres termes, la notion de *méthodes* (guides pour l'utilisateur) ne peut être commodément mise en oeuvre à l'aide de *métaprogrammes.* TOOLUSE (6) vise donc à fournir les moyens d'une assistance active pour la conception, l'implémentation, et l'évolution de logiciels. L'hypothèse de base est qu'une telle assistance doit être obtenue par la description du développement d'une manière formelle qui pourra donc être elle-même manipulée. Il est bien évident que la formalisation des développements passe par les étapes suivantes :

• étude et compréhension des méthodes de développement (7),

• formalisation du processus de développement (8),

• définition et réalisation d'un langage de développement (9),

• réalisation de l'environnement support.

Dans sa version actuelle, DEVA (le langage de développement) permet de spécifier des développement simples, rejoués à partir des exemples classiques. Un prototype de l'environnement est en cours d'achèvement.

## Le projet REPLAY

REPLAY est le projet compagnon de TOOLUSE. Son rôle principal est de mesurer l'impact des mécanismes et outils de manipulation de développement de logiciels dans un contexte de réutilisation de logiciels. Il est donc complémentaire de TOOLUSE où l'accent est mis sur les aspects *expression des développements.* Dans le but de promouvoir la réutilisation et les aspects évolutifs des développements déjà effectués, REPLAY concentre ses études sur :

• la composition de plans de développement, où interviennent à la fois les aspects réutilisation descendante et assemblage ascendant,

• le contrôle permanent des propriétés afin de montrer que les contraintes opérationnelles spécifiées dans le cahier des charges du logiciel à produire sont satisfaites (ou non).

## Les projets ESPRIT

L'équipe CPAO du Département d'Etudes et de Recherches en Informatique de CERT n'étant pas subventionnée, elle doit rechercher ses ressources financières auprès de divers contractants. Parmi ceux-ci, la Communauté Européenne a une place de choix depuis 1984. En effet, le financement de la continuation du projet SPRAC est assuré par la CEE avec les projets TOOLUSE et REPLAY. Le passage d'une entité nationale à une entité européenne a fortement influencé à la fois le contenu des recherches et la façon de travailler.

La constitution de deux consortiums responsables respectivement de TOOLUSE et de REPLAY a conduit à établir des liens privilégiés avec les laboratoires et industriels suivants :

• pour TOOLUSE
- en Irlande, le Trinity College à Dublin (TCD) (Prof. K. Ryan) et la société Generics (H.J. Kugler)
- en Belgique, l'université catholique de Louvain (UCL - Unité d'informatique) (prof. M. Sintzoff)
- en Allemagne, le laboratoire GMD à Karlsruhe (Prof. S. Jahnichen) et la société Biomatic à Freiburg (G. Koch)
- enfin, en France, la CISI-Enginierie à Toulouse (H. Horgen)

• pour REPLAY, CISI-Enginierie et UCL (Prof. E. Milgrom) sont des partenaires communs, avec en plus :
- en Grèce, la société ALPHA-SAI à Athènes (M. Gazoulidis)
- au Danemark, la société CRI A/S à Copenhague (J. Guldberg)
- en Belgique, la société Expert Software Systems à Gand (M. Huybrechts).

L'impact des relations privilégiées avec des laboratoires et des industriels étrangers est important, tant sur les plans purement techniques (échanges et transferts de technologies) que sur les plans personnels (adaptation à de nouvelles mentalités, vision plus européenne du travail, échanges de personnes). Les retombées positives du programme ESPRIT sont déjà très perceptibles.

## Autres contacts internationaux et industriels

Pour autant, les projets ESPRIT ne nous font pas perdre les liens forts qui existent avec d'autres équipes ou industriels. En particulier :

• l'INRIA à Sophia-Antipolis avec le projet GIPE (G. Kahn),

• l'université de Grenoble avec les projets LPG (D. Bert) et ADELE (J. Estublier),

• J.R. Abrial avec le démonstrateur de théorèmes B,

• le CNET avec le projet CONCERTO

• etc.

# ARCHITECTURE DE MACHINES LANGAGES & OUTILS POUR L'INTELLIGENCE ARTIFICIELLE

*SOL LEWITT*
*"STRUCTURE[illegible]" 1973-83, BOIS, PEINTURE A L'HUILE*
*INSTALLATION capc Musée d'art contemporain [illegible]*

NOTE DE PRESENTATION

Patrick Greussay, Patrick Salle





## "Langages et Outils pour l'Intelligence Artificielle" - Architectures de machines-langages

Ce pôle réunit actuellement toutes les grandes équipes françaises universitaires et industrielles de renommée internationale qui travaillent, d'une part, sur les langages LISP, PROLOG et leurs dérivés, les langages d'acteurs et les langages à objets, d'autre part, sur l'architecture de machines dédiées à ces langages.

C'est un pôle très actif et très productif qui voit apparaître de nouveaux projets menés par de jeunes équipes. Des produits sont commercialisés comme Lelisp, Prolog-II, l'éditeur Winnie et bientôt le coprocesseur Mali. Ils font partie d'un ensemble très important de logiciels qui pour beaucoup ont dépassé le stade du prototype.

Les liens du pôle avec l'industrie sont nombreux. Certaines équipes sont membres de laboratoires industriels renommés, Laboratoire de Marcoussis, Centre Scientifique IBM. D'autres sont à l'origine de la création de sociétés qui commercialisent leurs produits. Toutes ont des relations industrielles importantes qui permettent à ce pôle d'être actuellement présent dans quatre projets ESPRIT. Enfin deux équipes participent aux efforts de normalisation en étant membres de groupes : ISO, AFNOR et EULISP.

Du point de vue des thèmes, LISP est toujours l'objet d'études fondamentales : normalisation, sémantique et d'amélioration des techniques d'interprétation et de compilation. De nouveaux interprètes portables ont été développés : COMMON-LISP, MétaVlisp, SCHEME, PLASMA.



On peut noter également la permanence des recherches sur PROLOG avec la production d'interprètes et de compilateurs. Les études théoriques ont permis la définition du langage Prolog-3 qui introduit la notion de contrainte sur les clauses de Horn. Le problème crucial de la récupération mémoire a trouvé des solutions logicielles et matérielles. Une architecture multiprocesseur pour une exécution parallèle de PROLOG est à l'étude.

La communauté travaillant sur les langages à objets s'est fortement développée. Le modèle défini dans ObjVlisp unifie les notions de classe et de métaclasse et clarifie les concepts utilisés. Le rapprochement entre la notion d'objet et la programmation logique semble un thème prometteur avec les projets Eloise, Ciel et Glance. Le lien avec la simulation est également à l'étude dans plusieurs équipes.

La programmation parallèle fait partie des nouveaux enjeux avec pour cible les réseaux de sations et les multiprocesseurs. C'est un thème transversal que l'on retrouve dans les langages d'acteurs AL/1 et ABCL, les langages à objets LORE et objPive, et la programmation logique. C'est aussi l'étude de la programmation des architectures parallèles synchrones et plus récemment celle de l'approche connexioniste.

# COALA : Un projet de Calculateur Orienté Acteurs pour la Logique et ses Applications

COALA *MOTS CLES*
architecture multiprocesseurs
langages de cinquième génération
parallélisme et prolog
programmation par acteurs
unification

## Références

C. Percebois, I. Futo, I. Durand, C. Simon, B. Bonhoure, *"COALA : Un Calculateur Orienté Acteurs pour la Logique et ses Applications"*, Journées AFCET-GROPLAN : Architectures de Machines, Bigre + Globule n° 50, pp. 48-56, Gien, France, janvier 1986.

C. Percebois, I. Futo, I. Durand, C. Simon, B. Bonhoure, *"An Actor-Oriented Multiprocessor Architecture for PROLOG : COALA"*, First Italian Conference on Logic Programming, Genova, Italy, March 1986 (invited paper).

I. Futo, C. Percebois, I. Durand, C. Simon, B. Bonhoure, *"Simulation Study of a Multiprocessor PROLOG Architecture"*, First Italian Conference on Logic Programming, Genova, Italy, March 1986 (invited paper).

C. Percebois, I. Futo, I. Durand, C. Simon, B. Bonhoure, *"Résolution Parallèle de Sous-Buts Indépendants dans le Graphe de Connexion de R. Kowalski"*, Cinquième Séminaire sur la Programmation en Logique, pp. 553-570, Trégastel, France, mai 1986.

I. Durand, *"Un Modèle d'Interprétation Répartie pour une Architecture Multiprocesseur PROLOG"*, Thèse de Doctorat de l'Université Paul-Sabatier, Toulouse, France, octobre 1986.

C. Simon, *"Spécification et Simulation d'une Architecture Multiprocesseur PROLOG"*, Thèse de Doctorat de l'Université Paul-Sabatier, Toulouse, France, octobre 1986.

C. Percebois, I. Futo, I. Durand, C. Simon, B. Bonhoure, *"Simulation Results of a Multiprocessor PROLOG Architecture based on a Distributed AND/OR Graph"*, International Joint Conference on Theory and Practice of Software Development, TAPSOFT'87, pp. 126-139, Pisa, Italy, March 1987.



## De nouvelles architectures

Le projet COALA s'insère dans le cadre des calculateurs de la cinquième génération. L'objectif de ces calculateurs est de supporter efficacement le traitement symbolique en utilisant différentes techniques liées à la représentation des données et aux outils de contrôle, en particulier le parallélisme. Plus spécifiquement, l'étude a pour but de mettre en œuvre et de supporter le parallélisme inhérent au langage PROLOG, sans l'intervention du programmeur. Elle couvre trois aspects : la définition d'un modèle d'interprétation répartie pour le langage PROLOG, la spécification des mécanismes d'accès aux données et des outils de contrôle de l'architecture multiprocesseur et l'interconnexion des différents processeurs élémentaires.

## Prolog et le parallélisme

### LE MODÈLE D'INTERPRÉTATION RÉPARTIE

Le modèle d'interprétation répartie utilise le concept d'acteur et s'appuie sur le graphe de connexion ET/OU de R. Kowalski. Il exploite le parallélisme-OU dans la résolution et un parallélisme-ET/OU lié à un processus d'unification entre environnements.

Le graphe de connexion ET/OU est produit par une précompilation du texte source. Dans ce graphe, deux littéraux unifiables et de signe opposé sont reliés par un arc. Un arc, étiqueté par les liaisons des variables traduisant le résultat de l'unification des deux littéraux, exprime un résolvant potentiel.

Les différents résolvants sont construits et résolus en parallèle. Une étape de résolution consiste à sélectionner un arc et à construire le résolvant associé à partir de l'environnement étiquette de l'arc. Le parallélisme-OU est obtenu en sélectionnant en parallèle les arcs issus d'un même littéral et le parallélisme-ET en unifiant en parallèle l'environnement de l'arc choisi avec les environnements des arcs du résolvant.

Les unifications parallèles sur les arcs sont prépondérantes : elles induisent un parallélisme-ET/OU qui différencie notre modèle de ceux qui exploitent exclusivement le parallélisme-OU. Ce parallélisme-ET/OU entraîne une diminution de l'espace de recherche grâce à la détection anticipée de l'échec de certains littéraux qui deviennent non liés au graphe.

### LA STRUCTURE MULTIPROCESSEUR

Chaque processeur élémentaire possède une partie du graphe. Globalement, la mémoire peut être assimilée à un ensemble de mémoires locales ; le dialogue entre les processeurs étant assuré par des messages. Topologiquement, l'architecture proposée est composée de plusieurs processeurs élémentaires banalisés et interconnectés au réseau par une unité de communication qui assure le routage des messages. Chaque processeur ne dialogue qu'avec un nombre limité de voisins.

## Simulation du modèle

Un système complet de simulation de la structure multiprocesseur et du modèle a été réalisé. Il se compose de deux parties distinctes : un simulateur fonctionnel écrit en PASCAL et un simulateur système écrit en T-PROLOG.

Le simulateur fonctionnel, à partir du texte source PROLOG, construit le graphe de connexion correspondant, distribue le graphe à la structure multiprocesseur, puis exécute la question posée par l'utilisateur en simulant le comportement de l'interprète.

Afin de mesurer les performances de l'architecture, un deuxième simulateur, plus orienté système, fut nécessaire. Ce simulateur accepte en entrée la topologie du réseau d'interconnexion, la vitesse des canaux de communication et intègre les temps d'exécution des opérations élémentaires de l'interprète, obtenus à partir du simulateur fonctionnel. Il fournit en sortie différentes statistiques, en particulier le temps d'exécution du programme et la charge des différents processeurs élémentaires.

Ce système complet de simulation a permis de valider différents points spécifiques à l'architecture multiprocesseur COALA, comme par exemple le nombre de processeurs, la topologie du réseau d'interconnexion, la définition du processeur élémentaire et le domaine d'application de la machine. Les conditions de simulation reflètent presque exactement le comportement de l'interprète. Les applications simulées sont de taille raisonnable ; différentes organisations de machine ont été étudiées : point à point, grille, cube...

Les résultats de simulation ont montré qu'un gain substantiel est toujours constaté lorsque le nombre de processeurs augmente. De meilleurs résultats ont été obtenus pour les programmes PROLOG manipulant un grand nombre d'assertions. En pratique, il est possible d'envisager une topologie d'une centaine de processeurs ou plus. Ce nombre, raisonnable, rend la réalisation viable contrairement à d'autres approches présentant un nombre exorbitant de processeurs.

Concernant le réseau d'interconnexion, l'hypercube semble avoir le profil recherché. Ses performances sont intermédiaires entre la topologie point à point et la topologie grille et sa bonne régularité fait de lui un bon can-

Toulouse.

**Responsable scientifique**
R. Beaufils

**Organisme**
Laboratoire
"Langages et Systèmes Informatiques"
Université Paul-Sabatier
118, route de Narbonne
31062 Toulouse Cedex
☎ 61 55 69 46

**Participants**
C. Percebois
F. Guérin
A. Lamartinière
G. Orzati

didat pour une éventuelle maquette. La charge des processeurs élémentaires est correctement répartie et ce, de façon naturelle. Ainsi, aucun mécanisme de contrôle de flux n'est nécessaire.

La simulation a déterminé les unités fonctionnelles du processeur élémentaire. Initialement composé de deux unités, une pour l'unification, l'autre pour la résolution, le processeur élémentaire est maintenant organisé autour d'une seule unité de calcul. Ce choix a été validé par différentes mesures dynamiques montrant la disproportion des taux d'occupation des deux unités, quel que soit le nombre de processeurs élémentaires. Ainsi, la faisabilité d'un prototype a été démontrée avec un processeur élémentaire simple, banalisé et de faible coût.

## Extensions du modèle de base

### LE PARALLÉLISME-ET

La première extension du modèle concerne la détection dynamique de sous-résolvants indépendants. Il s'ensuit une forme de parallélisme-ET dans la résolution, en parfaite harmonie avec le parallélisme-ET/OU du graphe de connexion.

L'indépendance des littéraux d'un résolvant repose sur la connaissance des occurrences des variables du résolvant. Cette connaissance, mémorisée dans une table appelée table de dépendance, permet à un arc choisi par la résolution de construire dynamiquement la table du prochain résolvant. L'utilisation des tables de dépendance limite les unifications entre environnements du graphe et forme des groupes disjoints de littéraux lorsqu'ils existent.

Le modèle s'est avéré suffisamment souple, facilement modifiable. De nouveaux messages ont pu être ajoutés à la version initiale, sans perte d'homogénéité pour l'ensemble. Cette nouvelle version optimise jusqu'à 60 % le nombre d'arcs et de messages. Elle ne nécessite aucun algorithme complexe d'ordonnancement des littéraux, de retour en arrière ou de jointure des solutions partielles. Le modèle devient localement hiérarchisé et les solutions sont obtenues par produit cartésien des solutions partielles qui remontent dans le graphe.

Les premières simulations montrent que ce modèle est surtout exploitable pour des programmes déductifs-récursifs, déterministes ou pratiquement déterministes. Ce résultat est encourageant, car, pour cette classe de programmes, seul le parallélisme-ET peut conduire à des performances acceptables.

### LES APPLICATIONS PROLOG-BD

La deuxième extension du modèle résulte des bonnes performances constatées pour les programmes PROLOG manipulant beaucoup d'assertions. Ces résultats sont à l'origine d'une nouvelle version, actuellement en cours de développement et spécifique aux applications PROLOG-BD.

La présence de nombreuses assertions dans un programme PROLOG pose le problème de l'explosion combinatoire du nombre de messages de résolution et d'unification, à cause du parallélisme-OU.

Il est relativement facile de modifier le modèle initial afin de regrouper tous les messages en provenance d'un arc et à destination de plusieurs arcs appartenant à un même processeur. Il suffit de représenter tous les arcs d'une même partition sur le même processeur par un seul arc, qualifié d'arc virtuel, qui contiendra les environnements des assertions de cette partition.

La liste des arcs de résolution ne sera plus constituée que des adresses des arcs virtuels, ce qui diminuera sa taille et, par voie de conséquence, le nombre de messages émis. Ce nombre est maintenant fonction du nombre de processeurs de la structure multiprocesseur, mais en aucun cas de la taille des partitions, contrairement au modèle initial.

Les principaux avantages de cette méthode concernent donc le nombre de messages et le temps moyen de construction du futur résolvant. Un autre point positif réside dans l'algorithme d'unification lui-même puisqu'il ne s'applique que sur des variables et/ou des constantes ; les différentes assertions étant mémorisées dans une structure particulière appelée matrice de liaisons.

## Perspectives

### RÉALISATION D'UNE MAQUETTE

Après une phase de simulation qu'il est nécessaire de poursuivre afin d'affiner les résultats, nous souhaitons réaliser une première maquette à base de transputers.

Un premier travail d'optimisation de la version de base a été entrepris : les différentes structures de données de l'interprète ont été linéarisées et l'algorithme d'unification compilé, afin d'accroître les performances. Les premiers résultats de simulation, obtenus par extrapolation des mesures de la version de base, laissent espérer un facteur 5 en faveur de la version optimisée.

Parallèlement, nous travaillons sur la définition et la réalisation d'un noyau OCCAM pour l'écriture d'applications réparties. Ce noyau doit faciliter le portage de l'interprète sur l'architecture multitransputer.

### LES PRÉDICATS ÉVALUABLES ET LE MÉTA-CONTROLE

Sur le plan théorique, nos efforts actuels concernent l'introduction des prédicats évaluables PROLOG dans ce contexte de résolution parallèle, essentiellement les prédicats de contrôle (prédicat COUPE-CHOIX), les prédicats de manipulation dynamique de clauses (prédicats AJOUT et RETRAIT) et les prédicats de dialogue avec l'utilisateur (prédicats d'entrée/sortie de termes). Il est également envisagé d'introduire dans le modèle différents méta-prédicats pour guider et optimiser la résolution.

## Contacts internationaux

Par l'intermédiaire d'Ivan Futo que nous avons accueilli dans notre équipe pour une période de six mois, nous maintenons des contacts très étroits avec l'Institut pour la Co-ordination des techniques d'Ordinateur (SZKI) de Budapest. Notre collaboration concerne le méta-contrôle.

Irène Durand, qui a pris en charge le modèle d'interprétation répartie, termine actuellement un stage de recherche au sein de l'équipe du Professeur J. Minker à l'Université de Maryland. Cette équipe travaille également sur les problèmes du parallélisme en PROLOG.

I. Futo, C. Percebois, *"Les Concepts du Langage PROLOG et les Systèmes de Simulation"*, Revue d'Intelligence Artificielle, éd. Hermès, Vol. 1, n° 3, 1987, pp. 9-33, septembre 1987.

C. Percebois, *"Décomposition d'un Résolvant en Sous-Résolvants Indépendants"*, Journées AFCET-GROPLAN : Langages et Algorithmes, Rouen, France, novembre 1987.

C. Percebois, F. Guérin, G. Orzati, *"Parallélisme-OU et Assertions PROLOG"*, Septième Séminaire sur la Programmation en Logique, Trégastel, France, mai 1988.

C. Percebois, I. Durand, I. Futo, *"Parallel Execution of Independent Subgoals"*, International Symposium on Distributed Systems - Methods and Applications, IFAC-DIS'88, Varna, Bulgaria, June 1988.

# Pour une méthodologie de la programmation par objets

LANGAGES-OBJETS *MOTS CLES*
algorithmes parallèles
C++ (couche objet pour langage C)
CLOS (Common Lisp Object System)
Common Lisp (langage de programmation)
génie logiciel
intelligence artificielle
interface homme-machine
Loops
méthodologie de programmation
objets (systèmes)
ObjVLisp (langage objet)
programmation par acteurs
programmation par objets
prototypage
représentation des connaissances
SimTalk
simulation
SmallTalk (système objet)
types abstraits algébriques





## Références


*"An Evaluation Environment for Concurrent Object-Oriented Simulation"*, Jean Bezivin, SCS Multi'88, **Distributed Simulation Conference,** San Diego CA, USA, février 1988.

*"Types, Classes, Metatypes, Metatypes Classes: an Open-Ended Data Representation Model for Eu-Lisp"*, C. Queinnec et P. Cointe, soumis à la conférence Lisp 1988.

*"Towards the design of a CLOS Metaobject kernel: ObjVlisp as a first layer"*, P. Cointe, IWoLES (International Workshop on Lisp Evolution and Standardization), AFNOR-AFCET-INRIA, Paris, février 1988.

*"The ObjVlisp Kernel: A reflective Lisp Architecture to Define a Uniform Object-Oriented System"*, P. Cointe, **Meta-Level Architectures and Reflection,** North Holland, éditeurs : P. Maes & D. Nardi, Amsterdam, Pays-Bas, January 1988.

*"Some Experiments in Object-Oriented simulation"*, J. Bezivin, **OOPSLA'87** (Object Oriented Programming Systems Languages and Applications), Orlando FL, USA, octobre 1987, ACM Sigplan Notices, vol. 22, n° 12.

*"Metaclasses are First Class: The ObjVlisp system"*, P. Cointe, **OOPSLA'87,** Orlando FL, USA, octobre 1987, ACM Sigplan Notices, vol. 22, n° 12, pp. 156-167.

*"A Uniform Model for Object-Oriented Languages Using The Class Abstraction"*, J.-P. Briot et P. Cointe, **IJCAI'87,** Milan, Italie, août 1987.


## État de l'art : les objets, thèmes et variations

La programmation par objets est aujourd'hui une sorte de label revendiqué par des producteurs de logiciels très divers (intelligence artificielle, génie logiciel). Il en résulte une certaine confusion. Le but des travaux menés par l'équipe est d'abord d'étudier les techniques implémentatoires permettant de réaliser des systèmes objets pour en extraire les mécanismes de base et en déduire ensuite différentes méthodologies de programmation.

Le texte ci-dessous rend compte des résultats obtenus depuis 1986 et s'articule autour des principaux thèmes développés par la programmation par objets :

- résultats relatifs à l'étude des langages dérivés de Smalltalk-80 utilisant une taxinomie de classes et concernant la définition de nouvelles architectures objets pour Lisp : ObjVlisp et CLOS (Common Lisp Object System). En particulier développement de la technique des métaclasses comme outil d'auto-représentation des objets du langage et d'extensibilité de celui-ci,
- dans la lignée de Simula, étude de la contribution des langages à objets au domaine de la simulation et réalisation de la plate-forme d'expérimentation SimTalk,
- résultats relatifs à la contribution de la programmation par objets à la description d'algorithmes parallèles et au développement de la programmation dite par acteurs,
- étude de Smalltalk-80 comme modèle de réalisation d'interfaces de haut niveau (le MVC) et comme outil de prototypage rapide,
- étude des mécanismes annexes (contraintes, métaclasses, hiérarchie des parties, héritage multiple, démons, valeurs actives) permettant aux langages à objets d'aborder les problèmes de représentation des connaissances posés par l'intelligence artificielle,
- enfin, étude de l'intégration de la programmation par objets dans le monde des langages impératifs statiquement typés et étude de C++ comme outil de réalisation de nouveaux modèles de systèmes d'exploitation.

## Étude des langages à Objets à taxinomie de classes : le modèle ObjVlisp

Le travail de réflexion débuté par Jean-Pierre Briot sur les mécanismes d'instanciation et d'héritage dans sa thèse de 3e cycle puis repris par Pierre Cointe a abouti à un modèle unifié pour la sous-famille des langages à objets utilisant la classe comme système de taxinomie et comme modèle d'abstraction.

Ce modèle implémenté dans la dernière version d'ObjVlisp unifie les concepts de classe et de métaclasse : une métaclasse devient un objet à part entière, i.e., instance d'une vraie classe définie explicitement. En tant que classe, une métaclasse hérite normalement de ses sur-classes. Cette nouvelle théorie des métaclasses est décrite dans trois papiers qui ont été présentés à l'ECAI'86, à l'IJCAI'87 et à OOPSLA'87. Au niveau méthodologique, l'équipe étudie maintenant les possibilités offertes par la programmation par métaclasses, plus particulièrement en ce qui concerne les possibilités de paramétrisation des mécanismes d'héritage et d'instanciation dans le but de simuler de nouveaux systèmes objets.

L'unification métaclasse/classe nous a également suggéré une nouvelle définition du concept de variable de classe : nous proposons que les variables de classe définissant les connaissances globales à l'ensemble des instances d'une même classe soient définies explicitement comme les variables d'instance de la classe vue comme un objet. L'une des conséquences est la mise en évidence d'un nouveau schéma de représentation des connaissances utilisant le mécanisme de l'instanciation que nous pensons généralisable (voir l'exemple des polygones du papier IJCAI). On peut maintenant paramétrer les classes au niveau souhaité en n'utilisant que le seul concept d'abstraction/instanciation.

Enfin, notons que le modèle ObjVlisp est actuellement en cours de discussion par le groupe d'experts Eu-Lisp. Dans ce groupe, Pierre Cointe travaille avec Christian Queinnec à la définition d'un système de types extensibles, et non hiérarchisé, permettant le contrôle des entités allouées au niveau du bit et devant intégrer un système de classes à la ObjVlisp (voir le papier soumis à la Conférence Lisp 88).

## Contribution d'ObjVlisp à la réalisation d'un "Metaobject kernel" pour CLOS

Les potentialités des architectures réflexives (extensibilité, transparence, auto-description, introspection) donnent lieu à des recherches de plus en plus nombreuses dans les domaines de la programmation Lisp (3Lisp, MetaVlisp), de la programmation logique (FOL), des langages de frames (3KRS) et maintenant de la programmation par objets (voir le livre consacré au workshop

Jussieu

*Responsable scientifique*
Pierre Cointe

*Laboratoires*
LITP : Campus Jussieu, Tour 45-55 (bureau 209), 4, place Jussieu, 75005 Paris
Rank Xerox France : DRDBI, 12, place de l'Iris, Cedex 38, 92071 Paris La Défense
LAFORIA : Campus Jussieu, Tour 45-55, 4, place Jussieu, 75005 Paris
LIB : UBO-ENSTBr, 6, avenue V.-Le-Gorgeu, 29287 Brest Cedex

*Participants*
Jean-François Perrot
Jean Bezivin
Jean-Pierre Briot
Nicolas Graube
Alain Deutsch
Philippe Gautron
Robert Voyer

sur "Meta-Level Architectures and Reflections" organisé en octobre 87 à Alghero).

Jouant pleinement son rôle de laboratoire d'étude et de simulation, le système ObjVlisp nous a conduit à aborder les techniques des architectures réflexives. Le point de départ est encore le mécanisme d'instanciation et le mécanisme de régression à l'infini induit par les deux postulats :

- toute entité est un objet,
- tout objet est instance d'une classe, donc elle-même un objet d'après le postulat précédent.

Pour réaliser cette régression, une technique classique (du moins en Loops et Smalltalk) est de définir une boucle sur la racine du graphe d'instanciation, la racine se définissant comme sa propre instance. Contrairement aux autres systèmes objets, ObjVlisp donne la définition de la classe racine (Class) en ObjVlisp même. Cette solution demande la description du bootstrap et conduit à décrire précisément l'allocation et l'initialisation d'une instance-terminale, d'une classe et d'une métaclasse.

La description de ce boot-strap permet d'expliciter à l'utilisateur l'architecture du système des classes (le couple Class/Object), ensuite la technique de la programmation par métaclasses lui permet d'étendre celui-ci (voir le rapport de DEA de N. Graube).

Il se trouve que, parallèlement aux travaux sur ObjVlisp, le sous-comité CLOS en charge de spécifier le système objet pour ANSI-CommonLisp utilise lui aussi les architectures réflexives pour construire le "Metaobject kernel" de CLOS (voir le papier de Bobrow et Kiczales à IWoLES'88).

Nicolas Graube a entrepris de transformer le modèle ObjVlisp et sa machine virtuelle pour montrer que la technique des métaclasses ObjVlisp permet d'aboutir en plusieurs étapes au Metaobject kernel de CLOS dont il constituerait donc un sous-ensemble. Ce travail est la première partie de sa thèse consacrée aux architectures réflexives pour les langages à objets.

En parallèle, Pierre Cointe travaille sur un ObjVlisp "multi-bootstrappable" permettant d'intégrer les fonctionnalités de CLOS (les variables d'instances, les méthodes, les combinaisons des méthodes et les fonctions génériques deviennent des objets décrits par des classes). Il partagera avec Danny Bobrow une session sur les objets en Lisp lors du workshop sur l'évolution de Lisp et sa standardisation (voir le papier IWoLES).

## Langages à objets et simulation : SimTalk

Les problèmes abordés au LIB par Jean Bezivin sont principalement ceux de l'utilisation de différents paradigmes de programmation dans un contexte de simulation et porte principalement sur l'étude du paradigme objet dans certaines de ses relations avec le paradigme processus.

Cette étude est réalisée en prenant comme champ d'investigation le domaine des langages de simulation à événements discrets. Pour ceux-ci, une plate-forme d'expérimentation (SimTalk) a été construite au dessus de Smalltalk-80 dans laquelle plusieurs modèles de parallélisme, de synchronisation et de communications peuvent être utilisés et comparés.

L'idée de base consiste à considérer un processus comme un objet doté d'un fil de contrôle. Plusieurs modèles peuvent alors être étudiés. Dans le modèle client-serveur par exemple, en plus des processus ou objets actifs, il y a lieu de considérer également des objets passifs ou serveurs. L'accès à ces objets passifs peut s'effectuer selon différents protocoles. Contrairement à des langages comme Ada, CSP ou Occam qui ont réduit la dualité processus/moniteur à la notion de tâche et qui se servent de ce mécanisme pour prendre en compte l'agrégation, le point de vue de l'étude consiste à tirer avantage des relations "tout/partie" disponibles dans un langage à objets pour construire, de façon structurée, au-dessus d'un mécanisme primitif de concurrence (le fil de contrôle) et d'un mécanisme primitif de synchronisation (le verrou ou le sémaphore) des mécanismes évolués de structuration de système.

La notion de contrainte (au sens de Thinglab) n'est pas étrangère à ce travail dans la mesure où elle nous permet de prendre en compte des situations de groupes où plusieurs processus ont un comportement indépendant mais cependant lié (banc de poissons, escadrille d'avions, etc.).

## Programmation parallèle et langage acteurs : ABCL

Jean-Pierre Briot, lors de son séjour de deux ans au Japon dans le laboratoire de Akinori Yonezawa, a pratiqué intensément la programmation par acteurs (plus particulièrement le système ABCL : Actor Based Concurrent Language, voir sa présentation à OOPSLA'86). Cette pratique de spécification, d'implémentation et d'utilisation des langages acteurs l'a conduit à une réflexion sur les modèles d'héritage dans le contexte des langages à objets concurrents (voir le papier présenté à ECOOP'87).

A la suite de ces expériences avec des langages à objets concurrents (ABCL, mais également ObjPive, extension de ObjVlisp vers les processus de Bernard Serpette et Pierre Cointe, et Formes, présentés dans le livre OOCP édité par A. Yonezawa et M. Tokoro), nous débutons maintenant la modélisation d'un langage à objets parallèle basé sur une série de couches dont l'assembleur est une implémentation du modèle des acteurs de G. Agha. Ce projet va bénéficier du travail de réflexion sur les architectures réflexives pour assurer la cohésion entre ces différents niveaux.

L'équipe dispose déjà de prototypes écrits en Lisp et envisage dès maintenant l'implémentation sur des multi-processeurs (réseau de stations de travail, puis hypercube et transputers).

## Langages à objets comme outil de prototypage : Smalltalk-80

L'enseignement du langage Smalltalk-80 nous a conduit à étudier la technologie du MVC qui permet d'associer à un Modèle (en fait une hiérarchie de classes) une Vue (une hiérarchie de vues) et un Contrôleur (une hiérarchie de contrôleurs). Cette méthodologie rend compte des interfaces entre un objet informatique, ses représentations externes sur l'écran et ses interactions avec l'extérieur via les boutons d'une souris. Outre qu'elle permet de prototyper très rapidement des applications de haut niveau d'interactivité, son intérêt principal réside dans la description – en terme de hiérarchie de classes – des bibliothèques Smalltalk décrivant les différentes sortes de vues et de contrôleurs.

En fait, Smalltalk-80 permet non seulement la description du langage mais aussi celle de son environnement que l'utilisateur averti peut donc adapter à ses désirs. Par exemple : redéfinition des menus, modification du comportement des espaces de travail, définition de nouveaux browsers (flâneurs) ou de nouveaux inspecteurs. Nous pensons donc que le MVC doit faire partie des méthodologies du génie logiciel.

## Applications systèmes des langages à objets "à types abstraits"

Philippe Gautron est membre du groupe SOR (Systèmes à Objets Répartis) de l'INRIA. Dans ce cadre, il développe sous la direction de Marc Shapiro un système à objets répartis (SOS) pour le projet Esprit SOMIW. Cette recherche utilise intensément le langage C++ et a donné lieu à la réalisation d'un éditeur de liens dynamique permettant la migration d'objets (voir le papier Usenix 87). Cet éditeur de liens autorise une communi-

cation inter-sites fiable qui devrait permettre d'utiliser SOS comme une plate-forme d'expérimentation d'un univers multimédia distribué reposant sur l'approche objet.

## Étude des apports des langages objets à l'Intelligence Artificielle

L'arrivée de Jean-François Perrot au LAFORIA a donné un poids plus grand aux recherches sur l'utilisation des langages à objets en Intelligence Artificielle.

"Il a notamment lancé un programme d'expérimentation en Loops, dans le cadre du grant Ranx Xerox France", sur divers thèmes intéressant le Cemagref et le CEA. Le but est de se faire une opinion motivée sur l'utilité des structures plus complexes que les classes/instances (démons, valeurs actives, règles) qui fleurissent dans les langages destinés à l'IA, et sur les styles de programmation avec métaclasses et héritage multiple (travaux de Thierry Fuhs).

D'autre part, Robert Voyer a poursuivi ses travaux sur les structures d'objets dans les mécanismes d'inférence. Il a écrit un moteur en Smalltalk-80 et développe un système original en cours d'expérimentation.
Jacques Ferber développe le langage Mering-3, issu de ses précédents travaux et explore les architectures "multi-agents". Sa thèse d'État sera l'aboutissement de cette recherche.

## Activités annexes de promotion de la programmation par objets

En dehors de ces premiers résultats scientifiques, le groupe a participé activement à l'organisation de différentes manifestations consacrées au thème des objets. Nous retiendrons plus particulièrement :

- l'école d'été 86 de l'AFCET à Montréal : la méthodologie objet illustrée au travers d'un cours sur Smalltalk-80 (J. Bezivin, P. Cointe et J.-F. Perrot),
- création de la première conférence européenne sur la programmation par objets (ECOOP'87) organisée par l'AFCET à Paris en juin 87 et participation active à ECOOP'88 qui aura lieu en août à Oslo. Par ailleurs, on parle déjà de l'organisation d'une conférence jointe ECOOP/OOPSLA au Canada en 89...
- animation de différents séminaires industriels (IGL, Orsys, Cognitech, Bull, Iriam, Completive, Elf...) consacrés soit à la programmation par objets, soit au langage Smalltalk (80 et V). Pour ces animations, l'équipe a conçu - sous forme de transparents - deux supports de cours qui correspondent à une formation en cinq jours,
- participation aux activités de normalisation du langage Lisp et du système objet associé : Pierre Cointe est membre des groupes AFNOR, EU-LISP et ISO et suit les travaux du sous-comité CLOS du groupe ANSI-LISP.

## Réalisations et logiciels disponibles

**ObjVlisp**

Trois versions sont actuellement disponibles : en Le-Lisp 15.2 (Cointe), en Kyoto CommonLisp (Consel et Deutsch) et en InterLisp-D (Graube). Du point de vue de l'implémentation, Charles Consel et Alain Deutsch ont notablement accéléré l'accès aux méthodes et aux variables d'instances en réalisant un "tree-walker" et un cache local à chaque classe. Le tree-walker permet de compiler les chemins d'accès aux variables à l'instant de la définition d'une méthode, le cache permet de stocker l'adresse d'une méthode dans le graphe d'héritage après son premier appel. Nicolas Graube a réduit le nombre de primitives de la machine virtuelle en augmentant encore le caractère réflexif. Il a par ailleurs utilisé les métaclasses pour implémenter les valeurs actives de Loops en ObjVlisp.

**SimTalk**

Est disponible sur tout système Smalltalk-80.

**CLOS**

Le prototype de CLOS réalisé par Nicolas Graube est disponible sur la version Xerox de CommonLisp (machine 1186).

## Contacts Internationaux

L'équipe travaille en collaboration étroite avec plusieurs équipes étrangères :

**Xerox PARC** : Danny Bobrow et Gregor Kiczales (CLOS).
**EuroParc** : Tom Moran et Austin Henderson (interface : NoteCard, HyperCard).
**Tokyo-IT** : Akinori Yonezawa (ABCL).
**MIT Media-Lab** : Henry Lieberman (acteurs, multi-média).
**ParcPlace Systems** : Adèle Goldberg, Glen Krasner et Peter Deutsch (Smalltalk-80).
**Université libre de Bruxelles (VUB)** : Luc Steels et Pattie Maes (Computational Reflection, KRS).
**Université de Pise et Delphi Sa** : Giuseppe Attardi et Maria Simi (DCLOS, Omega).
**Université de Montréal** : Jean Vaucher et Guy Lapalme (Smalltalk-80, Plasma).
**Université d'Oslo** : Kristen Nygaard (Beta).



*"Two extensions to C++"*, P. Gautron et M. Shapiro, **Proceedings of the first C++ workshop USENIX,** Santa Fe NM, USA, novembre 1987.

*"TimeLock: A Concurrent Simulation Technique and its Description in Smalltalk-80"*, J. Bezivin, **WSC'87** (Winter Simulation Conference), pp. 503-506, Atlanta GE, USA, décembre 1987.

*"Applying the MVC Scheme to Simulation Software"*, J. Bezivin, **International Conference on Modelling and Simulation,** Karlsruhe, FRG, juillet 1987.

*"The transparency property of TimeLock, a Concurrent Object-Oriented Simulation Technique"*, J. Bezivin, **International Conference on Modelling and Simulation,** FRG, Karlsruhe, juillet 1987.

*"Inheritance Mechanisms in Object-Oriented Concurrent Languages"*, J.-P. Briot et A. Yonezawa (TIT), **ECOOP'87** (European Conference On Object-Oriented Programming), LNCS n° 267, Springer Verlag, éditeurs : J. Bezivin, P. Cointe, J.-M. Hullot et H. Lieberman, 1987.

*"The Formes System: A Musical Application of Object-Oriented Concurrent Programming"*, P. Cointe, J.-P. Briot et B.-P. Serpette, chapitre du livre intitulé : **Object-Oriented Concurrent Programming,** MIT Press, éditeurs : A. Yonezawa et M. Tokoro, Cambridge MA, USA, 1987.

*"Une présentation de Smalltalk-80 au travers de son environnement de Programmation"*,

P. Cointe, **Convention Informatique,** pp. 155-167, tome B, Paris, avril 1987.

*"The OBJVLISP Model: Definition of a Uniform, Reflexive and Extensible Object-Oriented Language",* J.-P. Briot et P. Cointe, **Advances in Artificial Intelligence-II,** North-Holland, éditeurs : B. Boulay, D. Hogg et L. Steels, Amsterdam, Pays-Bas, 1987.

*"Les langages à Objets : un Nouveau Style de Programmation",* P. Cointe, **La Recherche,** vol. 17, n° 248, pp. 1564-1567, Paris, décembre 1986.

*"Object-Oriented Concurrent Programming in ABCL/1",* A. Yonezawa, J.-P. Briot et E. Shibayama, **OOPSLA'86** (Object-Oriented Programming Systems Languages and Applications), Portland OR, octobre 1986, ACM Sigplan Notices, vol. 21, n° 11, novembre 1986.

# Boîte à Outils pour le Développement d'Applications d'Intelligence Artificielle

**LORE** ***MOTS CLES***
bases de données
Eloise (couche pour la programmation logique)
environnements de programmation
intelligence artificielle
Loops
Lore (langage objet)
parallélisme et objets
programmation déclarative
programmation par objets
programmation par règles
simulation symbolique
systèmes de contrôle
systèmes experts





## Références

*"La programmation par objets; le langage Lore"*, Lettre IA n° 6, publication des Laboratoires de Marcoussis, Décembre 1986.

*"Lore: un langage objet relationnel et ensembliste"*, Ch. Benoit, Y. Caseau, Ch. Pherivong, Actes des 3[e] Journées d'études sur les Langages Objets, Paris, Janvier 1986.

*"Lore et la Programmation par Objets"*, Ch. Benoit, J.-L. Giavitto, Rapport n° R.G.8.86, Greco de Programmation, Avril 1986.

*"Knowledge Representation and Communication Mechanisms in Lore"*, Ch. Benoit, Y. Caseau, Ch. Pherivong, Proc. of ECAI'86, Brighton, Juillet 1896.

*"The Lore Approach to Object-oriented Programming Paradigms"*, Ch. Benoit, Y. Caseau, Ch. Pherivong, OOPSLA'86, Poster Session, Portland (Oregon), Septembre 1986.

*"An Object-oriented language: Lore"*, A. Caseau, Proc. of HICSS'87, Kona, Hawaii, Janvier 1987.

*"La machine LISE"*, P. Dixneuf, Rapport de DEA, Université Pierre et Marie Curie (Paris VI), Juin 1987.

*"Lore 2.2, Guide Utilisateur"*, Rapport technique. L.1.0., Laboratoires de Marcoussis, Août 1987.

*"Étude et réalisation d'un langage objet: Lore"*, Y. Caseau, Thèse de l'Université de Paris-Sud (Orsay), Novembre 1987.

La programmation par objets constitue une méthode informatique extrêmement féconde de modélisation des objets du monde réel et de description de leurs interactions. Elle est particulièrement bien adaptée à l'étude des systèmes pouvant être schématisés par un ensemble d'éléments ayant chacun leurs caractéristiques propres et interagissant par échange d'informations ou d'instructions.

L'intérêt de cette approche pour la représentation et la manipulation des connaissances en intelligence artificielle a conduit les Laboratoires de Marcoussis à développer un langage nouveau dénommé Lore, constituant principal d'un environnement de programmation modulaire.

Ce langage se caractérise par son haut degré d'abstraction, toutes les notions utilisées étant représentées sous forme d'objets, et par la puissance de son mode de communication. De plus, il intègre un système de gestion des exceptions offrant ainsi à l'utilisateur un environnement complet d'aide à la programmation.

Le développement en cours d'un système d'inférences, appelé Eloïse, et d'un environnement graphique et interactif spécifique, permet dès aujourd'hui de faire de Lore un outil puissant bien adapté aux problèmes de :

- simulation symbolique : fonctionnement d'équipements industriels ou de matériel de transport;
- systèmes experts : surveillance de systèmes complexes tels que les turboalternateurs, gestion de réseaux, aide à la maintenance, problèmes d'ordonnancement d'ateliers flexibles, conception assistée par ordinateur, compréhension et reconnaissance de la parole;
- gestion de bases de données : assistance à l'utilisation de catalogues ou de documentations etc.

Lore est actuellement disponible, au stade préindustriel, sur des postes de travail Sun. Il est utilisé comme logiciel de base pour des applications civiles ou militaires avec des filiales du groupe CGE. Diffusé auprès d'universités, il sert de support d'enseignement et de recherche. Lore a déjà fait l'objet de plusieurs présentations internationales.

## Lore et la programmation par objets

Lore propose un modèle homogène pour décrire des mécanismes de représentation des connaissances issues des langages à objets classiques et des langages de schémas, mais également, un protocole de communication qui se place dans un contexte différent d'un contexte purement séquentiel et applicatif. C'est ce modèle que nous présentons dans les paragraphes suivants.

### OBJETS, CLASSES ET PROPRIÉTÉS

Lore est un langage à objets conçu pour la représentation des connaissances. Son originalité vient du fait qu'il repose sur un modèle puissant qui intègre les aspects de structuration propre aux langages à objets et les aspects relationnels propres aux langages de représentation de connaissances classiques. Il ne suffit pas que l'axiome "tout est objet" soit un point de dogme, définissant par sa seule existence la notion d'objet. Ce qui est important, c'est de disposer d'une définition bien définie d'un objet, dont *toutes* les entités du langage soient des représentations. Un objet n'est plus seulement la donnée d'un ensemble de caractéristiques statiques, les *attributs*, et de caractéristiques procédurales, les *méthodes.* Toute entité manipulée par le langage doit être un objet, décrit par le même modèle.

Pour ce faire, le modèle de Lore s'appuie sur les notions d'ensemble et de relation. Les ensembles sont le support des classes et les relations, celui des propriétés définies sur ces classes. De cette façon, un langage à objets est ainsi modélisé par la donnée d'une hiérarchie de classes (une classe = un ensemble), représentée par un *treillis* d'inclusion, et par des propriétés discrètes et fonctionnelles, représentées par des relations définies sur des produits cartésiens d'ensemble.

Cette homogénéité dans les concepts et cette structuration dans la description des caractéristiques et des comportements des entités sont essentielles, d'une part à la *lisibilité* et à la *modularité* des programmes, d'autre part à l'*extensibilité* du système. Cette approche nous offre des fonctionnalités originales comme la définition par sélection (regroupement d'objets vérifiant une ou plusieurs caractéristiques en un ensemble), ainsi qu'une méthodologie pour la résolution des conflits lors d'héritages.

### COMMUNICATION ET CONCURRENCE

L'implémentation classique de la passation de messages pour les langages à objets est l'appel fonctionnel ou procédural. La seule différence entre le fait d'invoquer une méthode et celui d'invoquer une fonction est que le premier implique un algorithme pour retrouver le code de la méthode tandis que le second a directement accès à l'adresse du corps de la fonction.
Or, la communication par passation de messages entre différentes entités du lan-

***Responsable scientifique***
Christophe Benoit

***Organisme***
Laboratoires de Marcoussis
Centre de Recherches de la C.G.E.
Route de Nozay
91460 Marcoussis

***Participants***
Françoise Carré
Yves Caseau
Patrick Dixneuf
Christophe Dony
Éric Hautemont
Alain Meller
Sylvain Picaud
Marc Porcheron
Pierre Raimond
René Velly

gage traduit à la fois un transfert d'informations et le déclenchement d'un traitement.

Un bon moyen de sortir du modèle de communication par appel/retour de fonction est d'offrir des primitives permettant à l'utilisateur de spécifier la façon dont l'évaluation d'un message doit intervenir. Les primitives de communication du langage doivent permettre de répondre à trois questions : quand, où et comment déclencher l'évaluation d'un message par l'objet receveur, c'est-à-dire décider du contexte d'exécution pour l'évaluation du message. Autrement dit, les primitives de communication du langage permettent à l'utilisateur de décrire la passation du contrôle de l'exécution entre différents objets et en particulier, le rôle de chacun des deux partenaires d'une communication : l'émetteur et le receveur.

L'envoi d'un message ne peut plus être confondu avec son évaluation, le receveur pouvant être un objet autonome auquel est associée une puissance de calcul traitant l'évaluation d'un autre message. On obtient alors un contexte d'exécution multiple où l'environnement objet est global et certains messages peuvent s'exécuter en temps partagé. Tous les problèmes de conflit d'accès et de partage de temps doivent être résolus par le système de messagerie. De plus, si le protocole de communication spécifié se place dans un contexte d'exécution répartie, l'environnement peut ne plus être global et le système doit gérer le convoyage des messages qui dépend alors de la topologie de la répartition des objets sur les processeurs.

En conséquence, pour un langage à objets, spécifier des primitives de communications qui ne se contentent plus de se substituer à un appel fontionnel, implique de mettre en place un véritable protocole de communication, soutenu par un système de messagerie. Il faut donc ajouter un niveau au langage à objets et s'intéresser à l'adéquation de ce niveau avec la machine hôte.

Le système de messagerie de Lore propose :

*UNE DICHOTOMIE ASK/SEND :* le rôle de l'émetteur est de déterminer le type de la communication qu'il désire faire : une requête (ask) ou une transmission (send). La primitive *ask* traduit une dépendance vis-à-vis d'un résultat. La primitive *send* traduit au contraire une indépendance totale vis-à-vis, non seulement d'un résultat, mais aussi de la bonne réception du message. L'objet qui émet ce message garde le contrôle et continue son traitement courant.

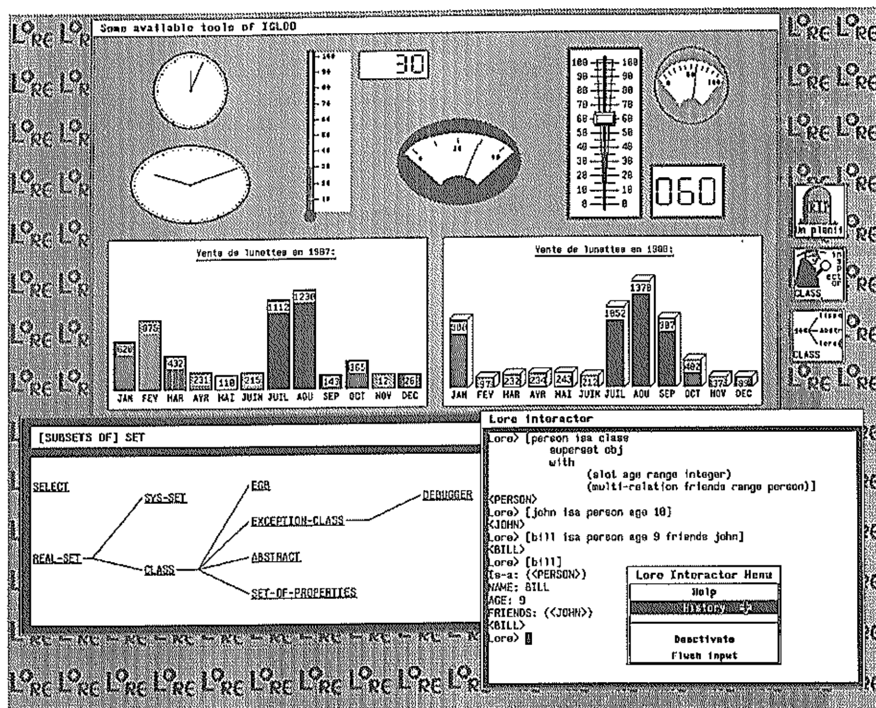
*DIFFÉRENTES FAÇONS DE DONNER LE CONTRÔLE AU RÉCEVEUR :* l'environnement d'exécution pour le traitement du message est-il créé pour l'occasion ou s'agit-il de l'environnement courant, c'est-à-dire celui du traitement effectué par l'objet émetteur ? D'autre part, lorsqu'il reçoit un message, un receveur est soit actif (au travail), soit inactif. S'il est inactif, il devient actif et effectue le traitement du message reçu. S'il est actif, plusieurs cas de figure sont possibles. Si le message a été envoyé de manière *asynchrone,* par exemple par la poste, il est stocké dans la boîte aux lettres associée au receveur et sera examiné plus tard. Si le message est envoyé de manière *synchrone,* par exemple par téléphone, le receveur a la possibilité de continuer à travailler (laisser sonner le téléphone), ce qui a pour effet de bloquer l'émetteur, ou bien de répondre, au risque de ne pouvoir effectuer son travail correctement. Lore définit donc une deuxième dichotomie entre les messages et distingué le mode immédiat (le message est traité dans le contexte courant) du mode bufferisé (un nouveau contexte d'exécution est créé).

## L'environnement de programmation de Lore

L'environnement de programmation de Lore est constitué d'une part d'un environnement graphique et interactif et d'autre part d'un environnement d'aide à la mise au point.

Devant la multiplication des matériels et des logiciels disponibles pour la construction d'interfaces interactives et graphiques, la nécessité s'est fait sentir de définir une base d'outils logiciels qui soient à la fois économiques, performants, d'une mise en œuvre simple et dont la portabilité soit envisageable sans trop de difficultés sur différents types de matériels.

L'architecture logicielle de l'environnement graphique et interactif est composée de deux couches distinctes. La première couche, appelée EGB, définit un environnement graphique de base. Il s'agit d'une bibliothèque d'objets assurant la gestion de fenêtrage, de tracé graphique et des moyens de saisie interactive que sont la souris et le clavier. La deuxième couche est une librairie d'utilitaires graphiques. Ces utili-




**Lore** *et son environnement de programmation.*

*"An experimentation of Concurrent Lore with MTCL: comparison with other languages"*, F. Carré, Tech. Report MADS TR-87.2, Esprit Project P440, Décembre 1987.

*"An Object-oriented Exception Handling System for an Object-oriented Language"*, Ch. Dony, Tech. Report, Laboratoires de Marcoussis, Decembre 1987.

*"Construction of Lore: from the model to the language"*, Y. Caseau, Tech. Report, Laboratoires de Marcoussis, Décembre 1987.

*"Eloïse: A Rule Oriented Programming Environment on Lore"*, P. Dixneuf, A. Meller, P. Porcheron. Tech. Report, Laboratoires de Marcoussis, Décembre 1987.

*"Eloïse: le système inférentiel du langage à objets Lore"*, P. Dixneuf, A. Meller, P. Porcheron. Rapport Tech., Laboratoires de Marcoussis, Décembre 1987.

taires sont développés entièrement à partir des objets disponibles dans la couche EGB.

La librairie est composée d'un ensemble de modules distincts regroupant chacun un certain nombre de fonctions logiquement liées selon le concept de boîte à outils, comme par exemple l'afficheur de graphes acycliques. Le rôle de cette librairie est de fournir un maximum d'outils prédéfinis afin, d'une part, d'alléger la tâche de programmation des interfaces graphiques et, d'autre part, de permettre une certaine uniformisation de la représentation et du style d'interaction.

L'élément principal de l'environnement d'aide à la mise au point est un système de gestion d'exceptions. Utilisable de façon standardisée par les modules du système et par les programmes utilisateurs, il offre une représentation en termes d'objets des exceptions et des *handlers*. Les *handlers* peuvent être associés à toute expression exécutable aussi bien qu'aux classes d'objets. Au sein d'un *handler*, il est possible de provoquer la reprise du calcul interrompu ou la terminaison de l'exécution de l'expression à laquelle le *handler* est attaché. Les outils de base de l'environnement de mise au point, destinés à la détection, la localisation et la correction des erreurs, i.e. des exceptions non traitées, utilisent avec profit ce système de gestion des exceptions. En particulier, l'outil d'examen de la pile d'exécution est écrit en Lore et utilise les possibilités du modèle avec reprise.

## Eloïse : le système inférentiel de Lore

Malgré ses nombreux avantages, un langage à objets *pur* est peu adapté à la description de comportements (heuristiques), de quantifications, de négations ou disjonctions de faits (au contraire d'un formalisme de règles de productions). D'une façon générale, un mécanisme d'inférences permet par contre une programmation déclarative des comportements des objets.

Le rôle d'un tel module intégré à un langage à objet, peut s'interpréter comme étant d'augmenter la capacité de réponse du système ainsi constitué face à des assertions ou des requêtes – le mécanisme d'héritage se charge déjà d'une partie de cette fonction. En effet, le système peut ainsi trouver les implications d'une nouvelle assertion (donnée par l'utilisateur ou engendrée par le moteur) grâce à l'exploitation de règles en chaînage avant. Et pour certaines requêtes, il peut exploiter des règles en chaînage arrière pour obtenir des réponses qui ne sont pas disponibles dans la base de faits. Il peut donc par exemple répondre à des questions du type :

Est-il vrai que ? *(chaînage arrière)*
Pourquoi est-il vrai que... ? *(explication du chaînage arrière)*
Qu'arrivera-t-il à... ? *(chaînage avant)*
Qu'arrivera-t-il si... ? *(chaînage avant hypothétique)*
Que devra faire l'opérateur pour que... ? *(chaînage avant + planification)*

Le système inférentiel Eloïse a été conçu dans l'objectif d'obtenir une intégration maximale avec le langage de programmation Lore afin d'offrir un outil homogène permettant de bénéficier, aussi bien des qualités des langages à objets, que des systèmes traditionnels à règles de production. Cette intégration réside avant tout dans le fait que le système inférentiel ne suppose pas de modèle particulier de représentation des connaissances et qu'il utilise donc directement la structure hiérarchique du monde des objets. La collaboration entre le langage à objets et le système inférentiel confère à l'outil Eloïse une puissance supérieure à des systèmes tels Art ou Kee, où ni la structuration objet du langage de représentation externe, ni sa disponibilité en tant que langage de programmation à part entière ne sont respectées. Cette collaboration rend Eloïse beaucoup plus proche des systèmes comme Loops ou Fork, eux aussi intimement liés à un langage à objets, dont ils respectent la structuration. Cependant, Eloïse possède un pouvoir d'expression supérieur à celui de ces deux systèmes et permet également de bénéficier de toutes les spécificités de Lore par rapport aux langages à objets sur lesquels sont bâtis ces deux systèmes.

Eloïse se caractérise par les éléments suivants :

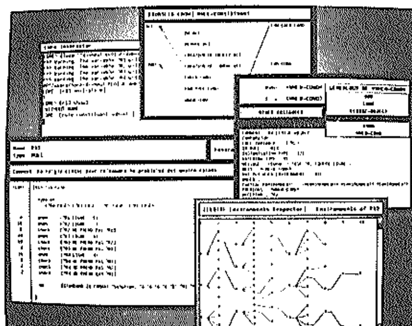
- toutes les entités Eloïse sont des objets Lore et le système est intégralement écrit dans ce langage;
- Eloïse utilise, pour vérifier la validité des prémisses des règles, les valeurs des attributs des objets du monde Lore;
- Eloïse modifie, en fonction des conséquences des conclusions des règles, les valeurs des attributs des objets du monde Lore.

Le module de chaînage avant d'Eloïse repose sur un moteur d'inférences du premier ordre autorisant non-monotonie et négations. Son langage de représentation permet l'écriture des règles sous forme déclarative, tout en conservant la puissance d'expression du langage Lore. Il permet également la structuration des règles en expertises et amas d'expertises. Son instanciation est assurée efficacement : seuls sont pris en compte, à chaque cycle, les changements intervenus dans la base de faits. Sa résolution de conflits est assurée soit par stratégies programmables, soit par

des méta-règles.
Le module de mise au point actuel offre à l'utilisateur d'Eloïse un environnement graphique et interactif permettant entre autres l'analyse pas à pas du déroulement des sessions du système et l'inspection graphique des objets ou des relations entre les objets.

## Réalisations logicielles disponibles

L'environnement de programmation Lore et le système inférentiel associé Eloïse sont des logiciels disponibles sur postes de travail Sun 3 et Common Lisp. La création d'un service de maintenance et d'aide à l'utilisateur, ainsi qu'une documentation technique complète, permet aujourd'hui un développement régulier des applications réalisées à partir de ces outils.



*Une application de simulation développée en collaboration avec Alsthom.*

## Perspectives futures

Les perspectives futures du projet "Boîte à Outils" s'articulent autour des axes suivants : (1) Développement de Lore et d'Eloïse, (2) Parallélisme et langages à objets, (3) Explorer les apports respectifs des technologies des bases de données et de la programmation par objets.

### 1 LORE ET ELOÏSE

Au-delà de l'enrichissement de l'environnement de programmation de Lore et l'adjonction de nouveaux modules à Eloïse (chaînage arrière, système de maintien de la cohérence), l'objectif global est de définir peu à peu une méthodologie de la programmation par objets.

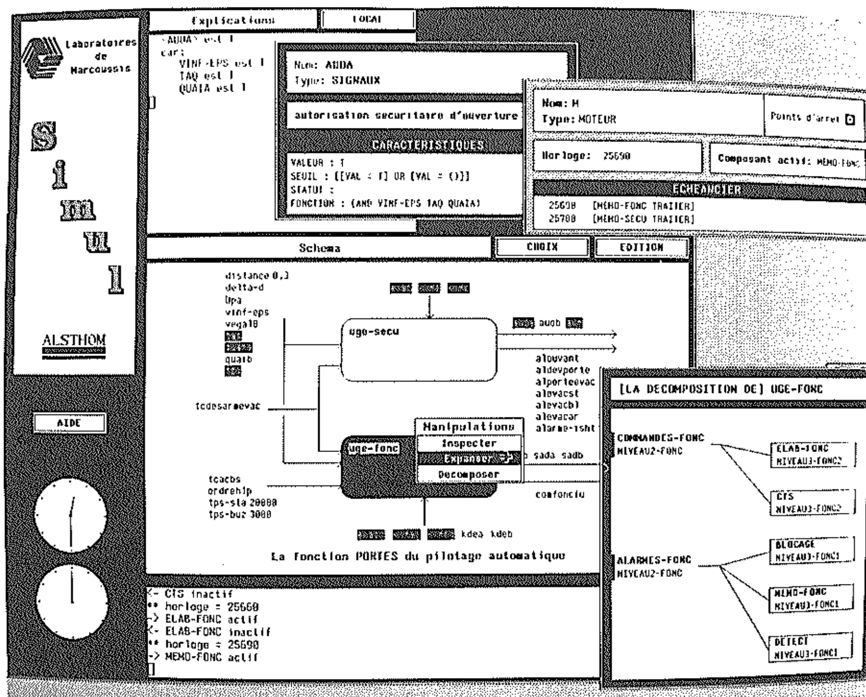
### 2 PARALLÉLISME ET LANGAGE A OBJETS

L'équipe est partie prenante au sein du projet Esprit P440 : Message Passing Architectures and Description Systems. L'objectif de notre participation est à terme de disposer d'un environnement d'exécution concurrent et réparti, intégré à un système de représentation des connaissances tels que Lore afin de supporter des applications du type : simulation, systèmes répartis, etc. L'élément principal de cet environnement est un Noyau Exécutif Concurrent comparable à un Operating System.

En collaboration étroite avec notre partenaire italien DELPHI, un modèle d'exécution parallèle a été spécifié. Il a été appliqué à Lore par une implémentation avec MTCL, extension concurrente de Kyoto Common Lisp développée par Delphi. Ce modèle d'exécution permet d'envisager la coopération puis le parallélisme dans les langages à objets, de façon nouvelle. L'ensemble des objets est considéré comme une donnée manipulée par les différents modules d'évaluations de Lore, ceci contrairement au classique modèle des acteurs.

### 3 BASES DE DONNÉES ET PROGRAMMATION DES OBJETS

La représentation des connaissances est un domaine central dans la construction des systèmes experts. Un intérêt grandissant concerne l'intégration de la technologie des bases de données avec la technologie des systèmes à base de connaissances (programmation par objets, inférence et déduction, raisonnement non monotone, etc.). Les objectifs de cette intégration sont notamment l'utilisation de grands volumes de données dans les systèmes d'IA et inversement la valorisation des bases de données existantes par des applications de type systèmes experts. Dans un premier temps, il s'agira de définir les apports spécifiques de Lore et d'Eloïse pour l'interrogation et la manipulation de base de données relationnelles.




**Éloïse** *et son environnement.*

# S3L
# Un interprète pour le langage Manens

MANENS *MOTS CLES*
théorie des algorithmes
CENTAUR (sémantique dénotationnelle)
ESTEREL (langage temps réel synchrone)
génération de programmes
intelligence artificielle
langage de spécification
MANENS (langage fonctionnel)
OLGA (outil de compilation)
programmation fonctionnelle
programmation relationnelle
RagTime (projet)
RAPTS
réutilisation des programmes
S3L (langage fonctionnel)
SETL
Syntax (méta-compilateur)
temps réel
théorie des algorithmes
types abstraits algébriques





Références

P. Boullier, *"Contribution à la Construction Automatique d'Analyseurs Lexicographiques et Syntaxiques"*, Thèse d'État, Orléans, 1985.

L. Cardelli, *"Implémentation de ML : une machine S.E.C.D. à Fermeture"*, Actes de la 13[e] école de Printemps du L.I.T.P., 1985.

J. Chailloux, *"Le-Lisp Version 15, Le Manuel de Référence"*, INRIA, 1985.

G. Cousineau, G. Huet, L. Paulson, *"The ML Handbook, Version 6.1."*, INRIA, 1984.

P. CIOSI, *"Unification en S3L : une Présentation Fonctionnelle"*, Journées Afcet-Groplan, Rouen, 1987.

F. Dalhaus, *"The Choice of Programming Primitives for SETL-like Languages"*, ESOP 86. Lectures Notes in Computer Sciences 213, 1986.

Darlington, Henderson, Turner, *"Fonctionnal Programming and Its Applications"*, Cambridge University Press, 1982.

P. Henderson, *"Functionnal Programming: Application and Implementation"*, Prentice-Hall International, 1980.

R. Hinley, *"Lambda-Calcul et Logique Combinatoire"*, Actes de la 13[e] école de Printemps du L.I.T.P., 1985.

J.-J. Lacrampe, *"Étude, Extension et Implémentation de MANENS"*, Thèse de Troisième Cycle, Université d'Orléans, Publications du LIFO, n° 86-3, 1986.

S3L est issu du langage MANENS, conçu par F. Le Berre et fondé sur la "théorie des Algorithmes" de L. Nolin. Dans cette théorie, tous les objets (fonctions et arguments), appelés algorithmes, sont de la même nature : ensembles d'un même domaine (TOUT) comportant les ensembles mathématiques et une classe de fonctions sur ces ensembles. Manens implante cette théorie. Tout objet du langage est donc à la fois ensemble et fonction. MANENS est un langage ensembliste et fonctionnel. C'est le matériel de départ du travail du groupe.

Une nouvelle version, implantée par J.-J. Lacrampe, tendait à en souligner encore le caractère fonctionnel :
- les constantes (ensembles mathématiques) deviennent des fonctions constantes qui sont leur propre résultat,
- les ensembles du langage peuvent désormais contenir des fonctions.

S3L est l'interprète noyau du langage résultant des réflexions du groupe MANENS. Le changement de nom tend à mettre l'accent sur le fait que les objets du langage, tout en conservant leur caractère fonctionnel, ne sont plus des ensembles, mais des "collections", n-uplets d'objets pour lesquels l'ordre est significatif, mais pas l'emboîtement.
- Le 0-uplet est la borne inférieure du domaine.
- Les 1-uplets représentent les scalaires des langages classiques.
- Les n-uplets ($n > 1$) dénotent le pluriel.

La version actuelle est écrite en Le-lisp version 15.2.

Les axes de travail du groupe sont les suivants :

1. Dans une perspective de validation du langage : écriture de la sémantique dénotationnelle de S3L et implantation du langage à partir de celle-ci en utilisant CAML.

2. Réalisation d'implantations plus performantes ou plus confortables.
- Implantation en C "à la main". La conception de S3L autorise une programmation sans variable. Sans imposer ceci à l'utilisateur, cette possibilité peut être exploitée au niveau interne pour construire un interprète performant.
- Implantation à partir de la sémantique dénotationnelle de S3L utilisant les outils de CENTAUR.
- Implantation utilisant les grammaires attribuées et comme outils SYNTAX et OLGA.

3. Étude de l'extension de S3L aux objets infinis. Par construction, des objets infinis font partie du domaine de S3L. Ils ne sont pas pour l'instant gérés par l'interprète. Techniquement leur prise en compte rend nécessaire l'introduction d'un concept d'équité dans l'évaluateur. Concrètement, le problème posé est celui de l'utilité de tels objets ou plutôt celui de la définition de primitives "intéressantes".

4. Étude de l'intégration dans S3L des programmations fonctionnelle et relationnelle. La construction théorique de S3L sous forme de domaine rend convergentes ces deux notions. Unification et résolution peuvent être abordées sous forme fonctionnelle. En liaison avec le traitement des objets infinis, S3L peut être étendu de façon à intégrer ces deux formes de programmation.

5. Utilisation de S3L pour la construction d'un système de gestion de bases de données. Fondé sur une approche ensembliste des bases de données relationnelles, il permettra de traiter de l'information disjonctive ou négative.

## Participation à un projet ESPRIT 2

Le groupe MANENS est engagé dans la préparation du Projet Esprit RAGTIME (Reusable Components and Automated Generation of Real-Time Implementations) à l'initiative de THOMSON-CSF.
RAGTIME se propose de construire un logiciel de haut niveau pour la spécification de systèmes comprenant du temps-réel et du parallélisme. Ses premiers objectifs sont :
- assister le développement de prototypes de haut niveau,
- permettre le passage des prototypes à des implantations efficaces et de bonne qualité.

Pour atteindre ces objectifs, Ragtime envisage d'utiliser :
- une machine transformationnelle basée sur CENTAUR,
- un langage de spécification à large spectre s'appuyant sur SETL,
- TYPOL pour formaliser la sémantique naturelle,
- une théorie transformationnelle utilisant RAPTS,
- ESTEREL pour la description formelle des systèmes temps-réel réactifs.

La participation du groupe MANENS à ce projet se ferait dans les trois directions suivantes :
- réflexion sur le langage de référence de Ragtime : SETL par comparaison avec S3L,
- étude de la commodité et de l'efficacité des outils Ragtime pour l'implantation de S3L en comparaison avec les autres implantations prévues,
- intégration des possibilités concurrentes de C-SETL dans les applications de S3L vers la programmation relationnelle et les bases de données.

orleans

**Responsable scientifique**
Jean-Jacques Lacrampe

**Organisme**
Dépt de Math
Université d'Orléans
Rue de Chartres
Domaine de la Source
45045 Orléans Cedex
☎ 38 63 22 16

**Participants**
Sophie Anglade
Andrée Chichereau
Pierre Ciosi
J.-J. Lacrampe
Dominique Laurent
François Le Berre
Marianne Ligou
Wahab Mekaouche

D. Laurent, *"La Logique des Partitions. Application à l'Information Disjonctive dans les Bases de Données"*, Thèse de Troisième Cycle, Université d'Orléans, 1987.

F. Le Berre, *"Un Langage pour Manipuler des Ensembles : MANENS"*, Thèse d'État, Université de Paris-7, Rapport L.I.T.P. n° 82-83, 1982.

M. Ligou, *"LTEE : Un Langage pour la Théorie Élémentaire des Ensembles"*, Thèse de Troisième Cycle, Université de Paris-6, 1987.

L. Nolin, *"Les Modèles Informatiques du Lambda-Calcul, Lambda-Calculus and Computer Science Theory"*, Lectures Notes, Springer-Verlag, 1975.

M. Jordan, D. Parigot, *"The FNC2 System: Users Guide and Reference Manual"*, Rapport Technique INRIA, 1987.

J.-T. Schwartz, *"On Programming: An Interim Report on the SETL project"*, 2nd Ed., Courant Institute of Mathematical Sciences, New York, 1975.

D. Scott, *"Data Types as Lattices"*, SIAM Journal of Computing, 4, 1976.

Groupe Langages INRIA, *"Manuel SYNTAX"*, Rapport Technique INRIA, 1986.

P. Weis, *"Le Système SAM"*, Thèse de Doctorat, Université de Paris-7, 1987.

# Une nouvelle méthode d'implémentation de langages de programmation

***Responsable scientifique***
Emmanuel Saint-James

***Organisme***
L.I.T.P. Université de Paris VI
4, place Jussieu
75230 Paris Cedex 05
☎ (1) 43 25 98 74

***Participants***
Charles Consel
Olivier Danvy
Marc Gengler
Pierre Jeanjean
Francis Kloss
Gérard Nowak

META *MOTS CLES*
environnements de programmation
génération de compilateurs
génie logiciel
machine abstraite/virtuelle
méta-compilation
méta-langage
méta-récursivité
méthodes pour la portabilité
unification

## Problématique

L'équipe Méta travaille sur une nouvelle méthode d'implémentation des langages de programmation, appelée *méta-récursivité,* consistant à utiliser le langage à implémenter comme étant son propre méta-langage. Cette technique pose plusieurs problèmes intéressants, et procure de nombreux avantages aux systèmes de programmation ainsi réalisés : fiabilité due à l'absence dans l'implémentation d'objets étrangers au langage, clarté des concepts issue d'une définition rigoureuse de l'outil d'implémentation, accessibilité totale par l'utilisateur des composantes du système, portabilité facilitée par l'absence de toutes considérations de bas niveau, pour ne citer que les traits les plus saillants. Les problèmes posés sont essentiellement le choix du bon sous-ensemble du langage spécifiant la machine virtuelle pour l'implémentation du langage complet, et les problèmes d'efficacité induits par le double niveau d'interprétation.

## Réalisations

Après une longue réflexion théorique sur le premier point durant l'année 1985, la réalisation de prototypes de tels systèmes a été effectuée en 1986, pour Lisp et les langages à Objets, sur différentes machines à base de 68200, 6800, 8086 et 6809. L'année 1987 a permis d'améliorer les performances des prototypes, et d'obtenir des systèmes concurrentiels par rapport à leurs homologues implémentés classiquement, grâce à des techniques originales d'interprétation, telles que l'exécution itérative de récursivités non-terminales, et l'élimination d'invariant de boucle dans les fonctionnelles d'ordre supérieur. Aujourd'hui, l'équipe propose deux systèmes, Méta-Vlisp et Méta-Objets, sur trois machines, Sun 3, PC et nano-réseaux, avec leurs manuels d'utilisation complets.

## Perspectives

Outre le portage sur d'autres machines, et l'amélioration des versions existantes, notamment au niveau de la communication avec les systèmes hôtes (Unix et MS-DOS), l'équipe compte à la fois montrer l'adaptation des systèmes à présent existants à différents problèmes de génie logiciel, et appliquer la méthode à d'autres langages de programmation, afin d'obtenir leur machine virtuelle idéale, et des techniques d'interprétation spécifiques. En ce qui concerne le deuxième point, les études portent sur les langages utilisant l'unification et la semi-unification d'ordre 2. En ce qui concerne le premier point, il s'agit d'une part de l'écriture de l'environnement de programmation (pas à pas *réversible* et compatible avec les interprétations itératives, tutorial intégré, désassemblage symbolique de la mémoire), et d'autre part la production automatique du compilateur à partir de l'évaluateur, par une étude théorique et pratique de l'*évaluation partielle.*

## Contacts industriels

Le Centre National de Documentation Pédagogique (CNDP) a passé commande à l'équipe Méta de langages de programmation implémentés selon les techniques développées. Meta-Vlisp et Méta-Objets opérationnels sur PC et Sun 3 et en version réduite sur MO5.

# Étude et réalisation d'un LISP de haut niveau

***Responsable scientifique***
Christian Queinnec

***Organisme***
L.I.T.P. Université de Paris VI
4, place Jussieu
75230 Paris Cedex 05
☎ (1) 43 25 98 74

***Participants***
F.-X. Testard-Vaillant
M. Quaggeto
P. Paroubek
V. Delacour
N. Seniak
B. Beaudoing
A. Deutsch

SCHISP *MOTS CLES*
intelligence artificielle
sémantique des langages
lisp (langage de programmation)
normalisation des langages
programmation par objets
sémantique dénotationnelle

## Le projet SCHISP

Lisp aura trente ans en 1988 et pourtant, sa normalisation via la très officielle Organisation Internationale de Standardisation (ISO) n'a débuté qu'en fin 1986. Cette évolution n'est que la conséquence naturelle de l'engouement croissant que lui porte l'industrie relativement au marché naissant mais prometteur que constitue l'Intelligence Artificielle.

La standardisation n'est pas un processus novateur en ce sens qu'elle ne doit normaliser que les pratiques existantes. Le cas particulier de Lisp constitue cependant un sujet de recherche complexe en raison :
- de la richesse de la communauté Lisp : plus de deux cents dialectes ont vu le jour et une dizaine (non compatibles entre eux) dominent le terrain. Tout existe mais un choix de traits harmonieux reste à opérer.
- La définition d'un ISO-Lisp ne peut être que le fruit d'un effort de formalisation qui n'a jamais été mené pour Lisp et qui, compte tenu de l'héritage mathématico-logique dont se réclament ses pratiquants, doit être d'un niveau supérieur aux normes actuelles (verbeuses et imprécises).

## Actualités et Perspectives

Dans le droit fil de ces tendances, l'équipe "Sémantique et Prospective des Langages Applicatifs" du LITP s'est lancée dans le projet SCHISP, projet d'étude et de réalisation d'un système ISO-Lisp de nouvelle génération. Les points forts de ce nouveau dialecte seront :
- une sémantique claire,
- une forte compilabilité,
- la nette séparation des environnements de développement et d'exécution,
- l'intégration de mécanismes pour la programmation par objets.

Outre sa participation à l'élaboration de ce nouveau dialecte qui sera proposé à l'AFNOR, l'équipe en étudie plusieurs réalisations.
- La première est une implantation en sémantique dénotationnelle qui servira de prototype de référence. Pour ce faire, sont en cours de développement des interprètes de sémantique dénotationnelle ainsi que des outils de mise au point (traceur, pisteur) et un interprète réversible (au sens de Lieberman).
Cette version de référence sert de laboratoire pour la mise à l'épreuve d'adjonctions (nouvelles formes spéciales) ou pour l'examen de variantes définitionnelles (macros, modules).
- La seconde est plus classique et vise la production d'un compilateur de qualité où sont plus particulièrement étudiés les points suivants :
- compilation fonctionnelle de qualité,
- typage fort optionnel et inférence de types,
- dérécursivation avancée,
- couche objet compilable,
- génération de bibliothèque d'exécution minimales,
- optimisation globale,
- intégration avec UNIX/C.

Par ailleurs, les besoins industriels d'aujourd'hui tels qu'ils peuvent être pressentis ne sont pas seulement de disposer d'un standard Lisp mathématiquement bien fondé, mais bien d'obtenir des compilateurs de haute qualité, conduisant à un code sûr, efficace et compact. La réunion des buts précédents est dans la droite ligne de la tradition française autour de Lisp. La réunion de tous ces traits est originale et devrait conduire à un intéressant prototype de compilateur.

## Relations Internationales

L'équipe participe aux travaux de normalisation à l'échelon français à l'AFNOR, européen dans le groupe EU-LISP et international à l'ISO.

L'équipe a aussi organisé conjointement avec l'AFCET, l'AFNOR et l'INRIA les Premières Journées Internationales sur Lisp, sa normalisation et son évolution (IWoLES'88).

# Programmation par acteurs

**PLASMA** *MOTS CLES*
ALOG (langage d'acteurs)
architecture parallèle
compilation par filtres
environnements de programmation
intelligence artificielle
LILA (machine virtuelle)
machine abstraite/virtuelle
parallélisme de contrôle
parallélisme des données
PLASMA++ (langage d'acteurs)
programmation logique
programmation par acteurs
programmation par objets
réseaux de machines

Le but du projet est la définition et l'implémentation d'un système complet et portable de programmation par acteurs. La portabilité est favorisée par la définition et l'implémentation sur divers sites d'une machine virtuelle avec laquelle nous réalisons toutes les implémentations des diverses composantes du système. Depuis la première implémentation et les études sémantiques du langage PLASMA, nous avons développé ce système qui comporte :

- une famille d'interprètes pour PLASMA et les langages d'acteurs ALOG, PLASMA++, AL1 dont PLASMA est le noyau et qui intègrent respectivement les programmations logique, à objets et concurrente.
- un système de trace et de mise au point,
- un compilateur du langage PLASMA,
- une machine virtuelle support des implémentations.

Les langages d'acteurs sont des langages apparentés aux langages applicatifs et issus de l'intelligence artificielle.
Dans un langage d'acteurs, une application logicielle est conçue comme un ensemble d'acteurs qui échangent des données par messages. Le contrôle s'exprime au moyen de ce seul mécanisme de transmissions de messages. L'acteur est une entité de calcul autonome. Il possède une mémoire locale et des connaissances, les acteurs qu'il connaît, et il réalise, à la réception d'un message qui s'effectue par filtrage, un traitement qui peut consister en trois tâches :

- création de nouveaux acteurs,
- transmission de messages à des acteurs et éventuellement à lui-même,
- modification éventuelle de son comportement.

Il en résulte deux sortes d'acteurs ayant un comportement distinct à l'égard des messages qu'ils reçoivent :

- les acteurs purs pour lesquels l'ordre d'arrivée des messages n'influe pas sur les résultats; ils peuvent donc être librement recopiés en instances traitant des messages de manière indépendante.
- les autres qui peuvent changer de comportement à chaque événement (réception d'un message); ces acteurs n'existent qu'en une instance unique et traitent séquentiellement les messages qui leur sont envoyés.

Dans ce modèle, tous les objets sont des acteurs, ce concept ne fait pas la distinction entre données et procédures.

Les interprètes des langages séquentiels PLASMA, ALOG et PLASMA++ sont réalisés sur la machine virtuelle LILA (Langage d'Implémentation des Langages d'Acteurs) implémentée sur SUN, VAX, MAC INTOSH.

## Recherches actuelles

L'apparition des machines à architecture parallèle n'a pas été accompagnée d'un développement comparable du logiciel pour utiliser efficacement ces nouveaux matériels. Un des problèmes fondamentaux est de trouver un langage adéquat pour mêler calculs parallèles et séquentiels.
Le modèle d'acteur a été développé afin de bâtir une théorie pour le calcul parallèle : si on peut associer à chaque acteur une puissance de calcul propre, l'ensemble des acteurs décrivant une application s'exécutera de manière concurrente.
Si le problème à résoudre est décomposé en parties indépendantes, traitées en parallèle, dont les résultats sont ensuite combinés, on parle de parallélisme de contrôle. Mais on peut également répartir un ensemble de données et appliquer en parallèle la même fonction sur chaque élément de l'ensemble. Il s'agit alors d'un parallélisme de données.
Le modèle d'acteur propose un moyen d'expression uniforme de ces deux formes de parallélisme.

Le langage AL 1 est un langage de programmation parallèle implémentant le modèle d'acteur et dont PLASMA est la composante séquentielle. Son implémentation nécessite des travaux simultanés sur une nouvelle définition de machine virtuelle MVR1 ainsi que des études sur un environnement de programmation distribué.

### LANGAGE AL 1

Comme en PLASMA, l'acteur a son environnement propre, mais pour lui allouer une énergie propre, on lui attribue un processus indépendant. La communication est asynchrone, ce qui nécessite un mécanisme de sérialisation des acteurs qui associe à chaque acteur une file gérée par un arbitre et qui contient les messages arrivés et en attente d'être traités.
Cette file est en fait le point d'entrée de l'acteur qui est référencé dans toute l'application par une forme contenant l'identification de la file et du processus. Ce mécanisme de sérialisation des messages a donné le qualificatif de sérialisé à un acteur pour lequel l'utilisateur exige explicitement une puissance de calcul; cet acteur sera généralement un acteur dont le comportement est modifiable.
En outre, lors de l'évaluation d'une forme d'expression du parallélisme contenant un acteur pur, une instance de cet acteur sera implicitement créée avec une puissance de calcul indépendante.
Ainsi à tout instant de son exécution, une application en AL 1 a ses données réparties entre divers acteurs et le contrôle est lui aussi distribué entre les différents acteurs actifs.

***Responsables scientifiques***
P. Salle, J. Vignolle

***Laboratoire***
Langages et Systèmes Informatiques

(LSI-ENSEEIHT),
UA 347 Université Paul-Sabatier
118, route de Narbonne
31062 Toulouse Cedex
☎ 61556965 - 61588383

***Participants***
J.-P. Arcangeli
M. Gandriau
A. Marcoux
C. Massoutie
C. Maurel
C. Pomian
P. Salle
M. Salomé
J. Vignolle

Les communications en AL 1 s'effectuent au moyen de trois types de transmissions :
- les transmissions non blocantes qui permettent de réaliser des calculs parallèles (FORK); l'acteur, site de la transmission, poursuit son exécution parallèlement à l'évaluation de cette transmission.
- les transmissions blocantes où l'émetteur attend une réponse pour continuer son exécution; elles sont implémentées par le mécanisme de capture de continuation en PLASMA.
- les transmissions de "type futur" que l'on pourrait qualifier de blocantes par nécessité. En effet, l'émetteur ne se met en attente du résultat de la transmission que lorsque ce résultat devient nécessaire à la poursuite de ses calculs.

Nous avons implémenté les mécanismes de changement de comportement, de priorité des messages et d'acteurs standard de parallélisme sur une version de la machine LILA étendue sous UNIX en utilisant les primitives de tubes et de fourches. Le pouvoir d'expression d'AL 1 a été mis en évidence par la programmation de multiples exemples classiques d'applications réparties. Il est alors apparu indispensable de concevoir une nouvelle machine considérée comme un réseau de machines virtuelles communiquant par messages et pouvant être distribuées sur divers types d'architectures matérielles.

### MACHINE VIRTUELLE MVR1

Ce réseau de machines est appelé MVR1. Chaque machine virtuelle est implantée sur un processeur physique déterminé et peut exécuter plusieurs acteurs en temps partagé avec les mécanismes classiques de priorité, section critique, etc. Un acteur peut envoyer un message à un autre acteur situé sur l'une quelconque des machines virtuelles.

Chaque machine virtuelle a une structure comparable à celle de la machine virtuelle LILA qui servait de support pour le développement des interprètes d'acteurs séquentiels. Il s'agit d'une machine de traitement de liste dotée d'un langage de type assembleur récursif capable de gérer simplement des continuations. L'avantage principal de ce langage est qu'il peut être transformé en code assembleur efficace ou en langage évolué par simple macrogénération. Sa syntaxe entièrement parenthésée permet un prétraitement par un langage applicatif pour réaliser des optimisations et tenir compte des caractéristiques propres à chaque machine cible.

### COMPILATION DE PLASMA

Nous étudions actuellement la construction d'un compilateur pour le langage PLASMA, composante séquentielle des langages d'acteurs, engendrant du code pour la machine virtuelle LILA. Cette compilation s'effectue suivant une démarche analogue à celle de la compilation de LISP, mais il s'y ajoute un ensemble de problèmes originaux comme la compilation du filtrage, des transmissions, des fermetures et des continuations. En revanche, à l'inverse de la majorité des dialectes LISP, PLASMA est un langage à liaison statique ce qui est un avantage pour la compilation.
Nous avons, dans un premier temps, défini une méthode de compilation des filtres et obtenu un compilateur de ces filtres, de structure équivalente à celle de l'interprète, par redéfinition des fonctions d'un interprète du filtrage. Ainsi, la compilation des divers types de filtres permet d'aborder la compilation de certaines structures comme la conditionnelle.
De façon à profiter des avantages qu'offre le langage, le compilateur est lui-même codé en PLASMA. Il peut ainsi se compiler lui-même.
Nous terminons actuellement la compilation des autres mécanismes et étudions l'intégration du compilateur dans l'environnement de PLASMA.

### ENVIRONNEMENT DE PROGRAMMATION

Les études sur l'élaboration d'un environnement de programmation du langage AL 1 s'appuient sur les systèmes de trace du langage PLASMA dans ses diverses implémentations et notamment en LILA sur MAC INTOSH. L'environnement de programmation du langage AL 1 est développé sur la machine MVR1 elle-même installée actuellement sur un réseau de stations SUN. La première phase entreprise est d'utiliser les multiples possibilités des stations SUN, notamment en matière de gestion de fenêtres à l'écran (modules SUN VIEW) afin de permettre à chaque acteur d'une application de disposer si nécessaire d'une fenêtre à l'écran qui servira d'interface entre l'acteur et l'utilisateur.

Dans la programmation concurrente, on rencontre de nouveaux problèmes liés à l'ordre partiel des événements. Ceci conduit à définir la nature des informations à conserver au cours d'une exécution et à étudier les différents choix de représentation de la trace.

### CIEL : CLASSES ET INSTANCES EN LOGIQUE

Découlant de l'expérience acquise avec ALOG, extension de PLASMA au traitement de la logique, CIEL est un PROLOG qui introduit la notion de classes et d'instances. Il présente à la fois les caractéristiques d'un langages à types polymorphes permettant une programmation générique et les caractéristiques d'un langage à objets avec un mécanisme de classes et d'instances et la notion d'héritage. La définition d'un objet CIEL comporte d'une part un ensemble de variables logiques typées locales, et d'autre part un ensemble de méthodes prédicatives définies par des clauses de HORN. Chaque variable logique est typée par le nom d'une classe ou par une variable logique, ce qui permet de définir des classes génériques paramétrées par d'autres classes. Toutes les méthodes d'une classe sont automatiquement typées par un algorithme d'unification qui détermine leur type le plus général.

## Réalisations logicielles

Les logiciels suivants sont des prototypes maintenus, diffusés avec manuel d'utilisation :

*LILA* Machine virtuelle pour l'écriture d'interprètes de langages applicatifs.

Version C sur VAX, SPS7, SPS9, SUN.

Version multifenêtre sur MAC+.

*La machine LILA est le support des interprètes.*

*PLASMA* acteurs séquentiels.

*ALOG* extension de PLASMA avec traitement de la logique.

*ACTOR* version parallèle de PLASMA utilisant processus et pipe-line UNIX.

*CIEL* Interprète disponible uniquement sur MAC+.

*Logiciels en cours de développement*

*MVR1* Machine virtuelle dérivée de LILA répartie sur un réseau de stations de travail SUN.

*AL 1* langage d'acteurs implémenté sur MVR1.

*PLASMA* le compilateur.

## Perspectives futures

Dans une première étape, l'équipe s'intéresse à l'implémentation du langage AL 1 sur la machine MVR1, en lui associant un environnement d'aide à la mise au point des programmes. Ensuite la machine virtuelle MVR1 sera implémentée sur une architecture multiprocesseur de type Transputer. Enfin, la compilation de PLASMA sera étendue à la version répartie.

PROLOG-III *MOTS CLES*
intelligence artificielle
langages de cinquième génération
prolog
prolog et parallélisme
résolution d'équations
systèmes de contraintes
systèmes experts

## Les fondements de Prolog III

Prolog a été initialement conçu pour le traitement des langues naturelles. Prolog III permettra un traitement intelligent de connaissances plus diverses, y compris celles qui font intervenir des données numériques. Essentiellement, il sera capable de traiter au niveau de l'unification :

- Les arbres infinis (créés par bouclage) qui seront munis de relations d'égalité et de différence.
- Les listes dotées de l'opération de concaténation **x. y** où **i**, la taille de la liste **x**, doit être précisée par la contrainte **x:i**, et doit être connue au moment de son écriture. Une liste est un arbre dont le sommet initial est étiqueté par le symbole $<>$. On la note $<a_1, a_2,...., a_n>$ où $a_1, a_2,...., a_n$ est la suite de ses fils immédiats. L'entier n, qui peut être nul dans le cas de la liste vide, est la longueur de la liste. La concaténation se définit simplement par $<a_1,...., a_m>.<b_1,...., b_n>=<a_1,.... a_m, b_1,...., b_n>$.
- Les nombres rationnels munis des opérations d'addition, de soustraction, de multiplication par une constante et des relations $=, =, \leqslant, \geqslant, >$. Ces nombres seront traités comme des arbres réduits à de simples feuilles.
- L'algèbre de boole complète avec toutes ses opérations et les relations $=$, $=$, et $\Rrightarrow$ (entraîne). Ici aussi les valeurs vrai et faux notées respectivement $\underline{1}$ et $\underline{0}$ sont considérées comme des feuilles d'arbres.

Dans ce Prolog, un programme est un ensemble de règles. Chacune est de la forme : $p_0 \rightarrow p_1 ... p_m$, **B** et se lit : sous réserve que l'ensemble **B** de contraintes soit satisfait, $p_0$ est vrai si $p_1$ et ... et $p_m$ sont tous vrais ou, dans une interprétation plus opérationnelle : pour exécuter $p_0$, il faut l'ensemble **B** des contraintes, puis exécuter successivement $p_0$ et ... et $p_m$.

Le mécanisme de base de la machine Prolog correspondante se résume en trois lignes :
(1) $(q_0\ q_1 ... q_n, A)$
(2) $(p_0 \rightarrow p_1 ... p_m, B)$
(3) $(p_1 ... p_m\ q_1 ... q_n\ A \cup B \cup \{p_0 = q_0\})$

La ligne (1) représente l'état de la machine à l'instant t, la ligne (3) l'état de la machine à l'instant **t+1** et la ligne (2) la règle utilisée pour ce changement d'état. Les $p_i$ et les $q_j$ sont des termes, A et $A \cup B \cup \{p_0 = q_0\}$ sont des ensembles de contraintes qui admettent au moins une solution.

## Récupération de mémoire en PROLOG III

Un récupérateur de mémoire s'est avéré indispensable pour PROLOG III. Cette contrainte nous a conduit à définir un interpréteur avec recopie utilisant deux piles :
- une pile de recopie,
- une pile de restauration.

A chaque unification, la règle utilisée, pour effacer un but, est dupliquée dans la pile de recopie. Cette pile contient en outre les sauvegardes des points de choix pour simuler l'indéterminisme.
La pile de restauration est constituée d'une suite de doublets <adr, cont> appelés souvenirs, où adr est l'adresse d'une cellule de la pile de recopie qui a été modifiée et dont le contenu de cette cellule avant cette modification. Pour le retour arrière, il suffit de remettre ces anciennes valeurs aux adresses indiquées pour actualiser la pile de recopie. Après l'effacement de certains buts, on peut mettre en évidence les seules structures utiles pour la suite de l'interprétation. La récupération de mémoire se charge donc de réorganiser la mémoire en compactant chacune des deux piles.
La récupération de mémoire se fait en trois étapes :

### 1. COLLISION DE SOUVENIRS ET CHAINAGE DE L'HISTORIQUE

Lors d'une même unification, une même cellule a pu être modifiée plusieurs fois. Seul le souvenir le plus ancien est utile pour le retour arrière. En parcourant la pile de restauration, d'une part, on élimine les souvenirs inintéressants et d'autre part, on met en place un chaînage de toutes les valeurs prises par une cellule de la pile de recopie pendant l'exécution du programme. Cet "historique" permet de "simuler" un backtracking lors de la deuxième étape.

### 2. ÉPURATION DE L'HISTORIQUE ET MARQUAGE DES STRUCTURES VIVANTES

Cette étape permet de marquer la résolvante courante ainsi que les résolvantes mémorisées dans les différents points de choix. On remarque lors du marquage que l'historique doit être épuré dans certains cas. Le marquage est lié à un point de choix en commençant par le point de choix le plus récent. On élimine de l'historique toutes les valeurs ne concernant pas le point de choix à partir duquel se fait le marquage. A la fin de cette étape, seules les cellules utiles des deux piles sont marquées. Il s'agit alors de compacter ces deux piles.

**Responsable**
Alain Colmerauer

**Organisme**
Dépt Mathématiques et Informatique
Université de Marseille II
13288 Marseille Cedex 2
☎ 91269000

**Participants**
Frédéric Benhamou
Jean-Marc Boï
Pascal Bouvier
Stéphane Grandcolas
Michel Henrion
John Kirman
Sophie Nabitz
Antoine Rauzy
Touraïvane

Marseille

### 3. COMPACTAGE SELON L'ALGORITHME DE MORRIS

Cet algorithme compacte la mémoire en mettant à jour les pointeurs. Parmi les différents algorithmes qui permettent d'effectuer ce travail, deux ont été retenus : MORRIS et JONKERS, d'une part, parce qu'ils conservent l'ordre linéaire des cellules vivantes après compactage, d'autre part, pour leur coût réduit en mémoire supplémentaire. Ils sont tous deux basés sur le même principe. L'algorithme de MORRIS a été préféré à celui de JONKERS parce qu'il est plus performant sur les structures manipulées par PROLOG III.

En voici une description rapide. On considère une suite de cellules A, B, C,..., qui pointent toutes vers une même cellule T. On retourne successivement chacun des pointeurs afin d'obtenir une liste (qui démarre de T) de toutes les cellules qui, initialement, pointaient vers T. L'information qui, initialement, se trouvait en T a été déplacée de façon à se trouver après le premier traitement, dans la dernière cellule de la liste. Dans un deuxième temps, quand on connaît la nouvelle position de T, soit T', on réactualise chacune des cellules de la liste en la faisant pointer vers T' et on retransporte l'information placée dans la dernière cellule, à sa place initiale soit T.

L'algorithme parcourt deux fois la zone mémoire : le premier passage se fait dans le sens du compactage, le second en sens inverse. Chacun ne s'intéresse qu'aux pointeurs dans le sens du passage courant et, par conséquent, ignore à la fois les données (non pointeurs) et les pointeurs en sens inverse. Le compactage ne se fait effectivement qu'au cours du second passage.

Pour l'implémentation, il est nécessaire de reconnaître :
- une cellule vivante,
- un pointeur,
- un pointeur retourné.

Pour des questions d'efficacité, on a choisi d'utiliser trois tableaux de bits extérieurs pour coder ces informations. On testera successivement des vecteurs de 32 bits pour trouver rapidement la prochaine cellule vivante.

D'autre part, on utilise un compteur, initialisé au nombre de cellules à récupérer (calculé lors du marquage) et incrémenté après le traitement d'une cellule vivante, qui permettra de connaître rapidement la nouvelle position de chacune d'elles.

Il reste maintenant à décider quand lancer le processus. Plusieurs possibilités sont offertes :
- lorsqu'une certaine borne limite de la pile de recopie est atteinte ou dépassée ;
- lorsqu'un certain pourcentage de la pile a été occupé, à partir du dernier appel du processus de récupération de mémoire ;
- par programme, c'est-à-dire par l'effacement d'une règle prédéfinie.

## Traitement des listes en PROLOG III

### 1. LES INTERPRÉTEURS DE PROLOG

a) Les interpréteurs à structure partagée (structure sharing) où l'on ne crée à chaque résolution que les variables apparaissant dans la règle appliquée (c'est à cette famille qu'appartient Prolog II, développé par le GIA).

b) Ceux qui fonctionnent par recopie de termes (structure copying) où l'effacement d'un but se fait à l'aide d'une copie de la règle à utiliser.

Le principal avantage des interpréteurs du type **1** est leur faible consommation de mémoire par rapport aux interpréteurs du type **2** qui en sont très gourmands. En contrepartie, les algorithmes intervenant dans les interpréteurs du type **1** sont plus complexes et plus lents que ceux du type **2** (les termes et variables sont plus difficiles d'accès).

Le nouvel interpréteur devant comporter par ailleurs des algorithmes d'unification déjà complexes, il était nécessaire de choisir le mode de fonctionnement **2**, quitte à implémenter un algorithme de récupération de mémoire (garbage collector) pour compenser la dépense excessive de mémoire.
La première maquette, écrite en Pascal, a permis de tester la faisabilité de ce nouvel interpréteur. Il était néanmoins intéressant de pouvoir en évaluer ses performances. Une autre maquette partielle de l'interpréteur Prolog, écrite en langage C, avec de nouvelles structures de données plus proches du mot-machine a donné de très bons résultats puisque sa vitesse d'exécution est de l'ordre de celle de Prolog II.

### 2. GESTION DE LA MÉMOIRE POUR LE NOUVEAU PROLOG

Le nouvel interpréteur nécessite dans sa version de base, l'utilisation de 2 piles principales et de 2 piles secondaires (4 piles secondaires dans la maquette courante, plus complète). Pour avoir une meilleure répartition de la mémoire entre ces piles, il est indispensable de les allouer dans une zone unique, en ne fixant aucune autre borne que celles de la zone et en permettant aux piles "intérieures" d'être translatables.

Cette gestion des piles permet de retarder l'appel au compactage mémoire, assez lent mais cependant indispensable.

### 3. NOUVELLES STRUCTURES DE LISTES POUR PROLOG

Les listes sont des structures très utilisées dans Prolog. Il est donc nécessaire d'optimiser les algorithmes permettant leur traitement en définissant de nouvelles structures et de nouveaux opérateurs que ceux existant dans d'autres dialectes de Prolog. Les opérations les plus fréquentes sur les listes sont les suivantes :

**a)** Accès au premier élément et au reste d'une liste (ces deux opérations de base sont communes à la plupart des interpréteurs prolog).
**b)** Concaténer deux ou plusieurs listes.
**c)** Accéder au $n^{ième}$ élément d'une liste.
**d)** Calculer la taille d'une liste.

On peut ajouter le point suivant : les chaînes de caractères sont indispensables à l'environnement. On souhaite, pour les manipuler plus aisément et bénéficier du traitement de liste, les assimiler à des listes de caractères.

Rappelons que les opérations des points **b, c, d** se programment récursivement dans les prologs classiques à l'aide des deux opérations du point **a.** Ce sont donc des opérations habituellement coûteuses en temps.

Nous avons défini de nouvelles structures de liste ainsi que les algorithmes permettant de traiter efficacement les points **b, c, d** au niveau de l'interpréteur Prolog.
- On note $<a_1,..., a_n>$ et on appelle liste de taille $n$ la suite ordonnée des éléments $a_1,..., a_n$.
- On note $<>$ la liste vide.

Soit "." (point ou concaténation) l'opérateur binaire associatif dont les opérandes sont des listes, défini par :
$<a_1, ..., a_n>.<b_1, ..., b_m> = <a_1, ..., a_n, b_1, ..., b_m>$
et, pour toute liste $l$ :
$<>.l = l.<> = l$.

On établit les restrictions suivantes :
- dans toute expression $a.b$, $a$ et $b$ doivent être des listes, et la longueur de $a$ doit être connue,
- pour toute équation $z = a.b$, il ne faut pas que $z$ "termine" $b$, afin de ne pas créer de liste infinie par bouclage.

### 4. RÉSOLUTION D'UN SYSTÈME DE CONTRAINTES SUR LES LISTES

***DOMAINE :***
Soit $C$ un ensemble de constantes.
Soit $V$ un ensemble de variables.
Soit $D = C \cup V \cup \cup L_n$ pour tout entier $n \geqslant 0$, les $Ln$ étant définis plus bas.
Soit $Ln$ l'ensemble des listes de taille $n$ de la forme :
$<a_1,...,a_n>$ pour $n \geqslant 0$, les $_i$ appartenant à $D$.

*TRAITEMENT DU SYSTÈME :*
On note $l : n$ la contrainte imposant à $l$ d'être une liste de taille $n$.
Soit $S$ un ensemble de contraintes de la forme $l1 = l2$ ou bien $l1 : n$, avec $l1$, $l2$ des éléments de $D$, $n$ un entier positif ou nul.
Simplifier le système $S$ consiste à le transformer en système $S'$ équivalent vérifiant la propriété suivante :

Les équations de $S'$ sont de la forme $v = l$, avec $v \varepsilon V$ et $l \varepsilon D$, où la variable $v$ n'apparaît en membre gauche dans aucune autre équation de $S'$.

On transforme le système $S$ en $S'$ en effectuant les opérations suivantes :
- $S'$ est vide au départ.
- On enlève de $S$ les contraintes de la forme $l : n$, en vérifiant ou en imposant que $l$ soit une liste de taille $n$. A ce moment, les restrictions énoncées plus haut doivent être respectées.
- On passe ensuite aux équations : tant que $S$ n'est pas vide, on lui enlève une équation $l1 = l2$ et on la traite (paragraphe suivant).

### 5. RÉSOLUTION D'ÉQUATIONS SUR LES LISTES

On ne s'étendra pas sur le traitement de ces équations. On ne donnera qu'une vague idée des algorithmes permettant le traitement des principaux types d'équations. Si l'équation à traiter est de la forme :
- $a.b = u.v$ avec $a$ une liste de taille $n$, $u$ de taille $m$, avec $m = 0$ et $n = 0$ (si par exemple on a $n = 0$ on a en fait à résoudre l'équation $b = u.v$).

On impose ou on vérifie - selon qu'ils sont libres ou pas -, que $b$ et $v$ sont des listes.
Si $n = m$, on ajoute à $S$ les équations $a = u$ et $b = v$ sinon supposons $n < m$ (si $n > m$, on traite l'équation $u.v = a.b$).
On introduit la variable $f$, à qui on impose d'être une liste de taille $m - n$.
On ajoute au système $S$ les équations :
$u = a.f$
$b = f.v$
qui seront traitées plus tard.
- $<a_1,...,a_n> = <b_1,...,b_m>$
on vérifie que $n = m$, puis on ajoute à $S$ les équations $a_i = b_i$.
- $<a_1,...,a_n> = a.b$
on ajoute à $S$ l'équation $<a_1,...,a_n>.<> = a.b$ (on se ramène en fait au premier cas).

### 6. STRUCTURES INTERNES UTILISÉES

On utilise 2 structures internes pour représenter les listes en mémoire :
- La structure représentant la liste d'éléments contigus $<a_1,...,a_n>$, est un doublet formé de l'entier $n$ et d'un tableau de pointeurs sur les éléments $a_i$.
- Une autre structure qui représente l'expression $a.b$. C'est un doublet de pointeurs sur les listes $a$ et $b$.

## Algorithmes de résolutions d'équations sur les arbres et les listes

Il s'agit de prolonger l'algorithme classique de résolution de systèmes d'équations et d'inéquations dans l'algèbre des arbres, proposé par Alain Colmerauer (Équations et Inéquations sur les Arbres, 1984) afin de pouvoir manipuler plus aisément certains de ces arbres, les listes.

### 1. STRUCTURE

Nous nous sommes placés dans l'algèbre des arbres dans laquelle nous avons introduit les opérations "." et "$L_n$", $n > 0$, respectivement binaire et $n$-aires. "$b$" étant une opération binaire particulière de notre algèbre, et $a_1,...,a_n$ $n$ arbres de son domaine, l'arbre $ba_1ba_2...ba_nL_0$ est une liste de longueur $n$ ($L_0$ est la liste de longueur nulle).
- L'opération de concaténation "." fait correspondre à 2 arbres $A$ et $B$, $A$ de la forme $ba_1ba_2...ba_nL_0$, l'arbre $.AB = ba_1ba_2...ba_nB$.
- Les opérations d'assemblage "$L_n$", $n > 0$, construisent à partir de $n$ arbres $a_1,...,a_n$ l'arbre $L_na_1...a_n = ba_1ba_2...ba_nL_0$.

### 2. SYSTÈMES D'ÉQUATIONS

Si $F$ est l'ensemble des symboles fonctionnels associés aux opérations de notre algèbre, et $V$ un ensemble infini de variables, un terme $t$ est une suite finie d'éléments juxtaposés de $F \cup V$. $X$ étant une affectation de l'ensemble des variables, et $t$ un terme quelconque :
- soit $t$ est réduit à une variable $x$, et $t/X = x/X$,
- soit $t = ft_1...t_n$, $f = .,$ et $t/X = ft1/X...t_n/X$ si $t1/X,...,t_{n-1}/X$ et $t_n/X$ sont définis,
- soit $t = .uv$, et $t/X = .u/X^v/X$ est une liste de longueur finie et si $v/X$ est défini.

Dans tout autre cas, $t/X$ n'est pas un élément du domaine de notre algèbre.

Un système d'équation $S$ est composé d'un ensemble fini d'équations plongé dans un ensemble d'affectations $C$, son contexte, tel que quel que soit $X$ une affectation de $C$, et quel que soit $t$ un terme figurant dans $S$, $t_{/X}$ est un élément du domaine de notre algèbre.

Pratiquement, nous utiliserons pour décrire nos contextes, des contraintes de longueur de la forme $Liste_k(t)$, $k > 0$. Une affectation $X$ satisfait la contrainte $Liste_k(t)$ si $t_{/X}$ est une liste de longueur $k$.

Un système d'équations est donc un ensemble d'adéquations et de contraintes de longueur. Afin de conserver à notre algorithme sa simplicité, nous exigerons que pour tout terme $.uv$ figurant dans un système d'équations $S$, il existe $k$, $k > 0$, tel que, soit $u$ débute par le symbole fonctionnel $L_k$, soit $S$ contient la contrainte $Liste_k(u)$.

### 3. ALGORITHME DE RÉDUCTION

Notre but est de déterminer si un système d'équations $E$ est soluble ou non, et s'il est soluble de mettre en évidence ses solutions. Pour cela, notre algorithme tente de produire un système d'équations sous forme "réduite" $R$, équivalent à $E$ au moins sur l'ensemble des variables figurant dans ce dernier. S'il y réussit, $E$ est soluble et nous pouvons décrire ses solutions à l'aide de $R$, sinon $E$ est insoluble.

$R$ est un système d'équations dont, d'une part, les membres gauches d'équations, et d'autre part, les termes figurant à l'intérieur de ses contraintes de longueur, sont des variables distinctes. A toute affectation $X$ des variables de $V$ qui ne figurent pas en membre gauche d'équation, satisfaisant les contraintes de longueur de $R$, correspond une affectation $Y$ unique des variables figurant en membre gauche d'équation telle que $X \cup Y$ soit solution de $R$.

### 4. EXEMPLE

Nous nous limiterons à des systèmes d'équations dans lesquels le symbole fonctionnel "$b$" n'apparaît pas, et pour tout terme de la forme $.st$, soit $s$ débute par un symbole fonctionnel "$L_k$", soit $s$ est une variable (on peut facilement modifier un système d'équations quelconque afin qu'il satisfasse ces restrictions). Nous écrirons alors $s.t$ le terme $.st$ sans risque d'ambiguité.

Soit à réduire le système d'équations :
$E = \{fax_1{=}fx_2L_3abc,\ y_1.y_2{=}z_1.L_4abcd,\ Liste_5(y_1),\ Liste_2(z_1)\}$
Notre algorithme produit le système d'équations soluble
$R = \{Liste_5(y_1),\ Liste_2(z_1),\ Liste_3(u),\ x_2{=}a,\ x_1{=}L_3abc,\ y_1{=}z_1.u,\ u{=}L_3abc,\ y_2{=}L_1d\}$
équivalent à $E$ sur l'ensemble de variables $\{x_1, x_2, y_1, y_2, z_1\}$.

## Traitement des contraintes numériques

Nous ne nous intéressons qu'aux contraintes numériques linéaires, c'est-à-dire à une suite d'équations et d'inéquations de la forme :
$b + \Sigma_i a_i x_i = 0$,
$b + \Sigma_i a_i x_i \geqslant 0$ ou,
$b + \Sigma_i a_i x_i > 0$,
où les $x_i$ sont des variables de type arithmétique, les $a_i$ et b des coefficients numériques. La méthode adoptée pour vérifier qu'un système de contraintes arithmétiques est soluble ou pas, repose sur l'algorithme du Simplex de G.B. Dantzig, avec traitement des cas de dégénérescences.

Le traitement des contraintes de type = d'une part, et la propagation des erreurs d'arrondis sur les flottants provenant des calculs propres au Simplex d'autre part, nous ont amené à restreindre notre domaine de travail aux nombres rationnels. En conséquence, une arithmétique fonctionnant en précision parfaite a dû être écrite.

### 1. L'ARITHMÉTIQUE

Pour des raisons d'efficacité et d'occupation mémoire, le codage des rationnels en développement en fractions continues a été abandonné au profit d'un codage plus classique sous forme d'un couple d'entiers (numérateur et dénominateur). Une première réalisation a permis de constater que pour certains problèmes, la taille des nombres entiers disponibles sur machine s'avérait tout à fait insuffisante ; l'arithmétique a donc été étendue aux entiers non bornés.

Plusieurs structures pour le codage des différents types de constantes numériques ont été étudiées, et c'est finalement par une suite contiguë de mots machine de 32 bits que les constantes ont été représentées. Ce codage présente l'avantage de manipuler des structures simples, conformes à celles utilisées par l'interpréteur, il permet de minimiser la place occupée par les entiers et les rationnels, surtout lorsqu'il s'agit de petits entiers (inférieur à $2^{24}$) qui restent de loin les plus fréquents. Enfin, c'est avec ce codage que nous avons jusqu'à maintenant pu réaliser les meilleures performances pour les calculs arithmétiques.

### 2. LE TRAITEMENT DES CONTRAINTES

Initialement, le système **S** de contraintes est vide. Chaque fois que l'on ajoute une contrainte **c** au système **S**, on applique au système $S \cup \{c\}$ une suite de transformations qui produit un système équivalent **S'**, pour lequel on peut décider s'il est soluble ou pas. Les contraintes arithmétiques de type = ne faisant pas l'objet d'un traitement particulier par rapport aux contraintes du même type sur les arbres, leur résolution se fait au niveau de l'interpréteur.

On peut distinguer quatre phrases principales dans ce traitement :
1. mise sous forme normale des contraintes,
2. réduction du système de contraintes **S**,
3. réduction du sous-système $S^+$ de **S** (processus Simplex),
4. élimination des variables figées.

#### *a. MISE SOUS FORME NORMALE*

Les inéquations du type $\geqslant$ et $>$ sont traduites par une équation de la forme
$b + \Sigma_i a_i x_i = u^+$
où $u^+$ est une variable d'écart introduite, astreinte à être non négative. Si l'inéquation est du type $>$ on produit aussi la contrainte $u^+ \neq 0$.

La forme normale d'une contrainte est $x = r$, où
(1) x est une variable qui ne figure pas dans r,
(2) r une expression arithmétique simplifiée,
(3) x » y pour toutes les variables y de r ("»" étant la relation d'ordre établie sur les variables au niveau de l'interpréteur avec la restriction suivante : x » y si x est une variable arithmétique et y une variable arithmétique astreinte à être non négative).
Par exemple, la forme normale de la contrainte $2x + 3y - 2 \geqslant 0$ sera
$x = -3/2y + 1 + 1/2u^+$ si x » y ou
$y = -2/3x + 2/3 + 1/3u^+$ si y » x.

Chaque contrainte est alors mémorisée sous sa forme normale (équation) et codée dans l'espace des règles comme une suite de doublets adresse-variable, adresse-coefficient (les monômes dont le coefficient est nul ne figurent pas dans l'expression).

#### *b. RÉDUCTION D'UN SYSTÈME DE CONTRAINTES*

Un système **S** de contraintes arithmétiques est sous forme réduite s'il est de la forme
$\{x_1 = t_1, \ldots, x_n = t_n\}$
et s'il satisfait aux conditions suivantes :
(1) toutes les variables de **S** sont de type arithmétique,
(2) les contraintes de **S** sont sous forme normale,
(3) les $x_i$ sont des variables distinctes d'une contrainte à l'autre,
(4) le sous-système $S^+$ de **S** contenant les contraintes $x_i^+ = t_i$ avec $x_i^+$ non négative peut se mettre sous forme réduite dans la structure des variables astreintes à être non négatives.

Réduire le système $S \cup \{c\}$ où **S** est sous forme réduite et **c** une contrainte du type $s = t$, produite au cours de l'unification, c'est substituer dans s et dans t chaque occurrence de variable par le membre droit de l'équation qui définit cette variable dans **S**, tant que cela est possible, et mettre l'équation obtenue sous sa forme normale **c'**. Dans le cas où le système initial de contraintes contient des inéquations du type $\geqslant$ ou $>$, cette réduction vise à éliminer les variables non astreintes à être non négatives.

L'équation obtenue **c'** peut avoir quatre formes :
(1) **c'** est trivialement insatisfaisable ($k = 0$, k constante non nulle), $S \cup \{c\}$ n'a pas de solution,
(2) **c'** est trivialement satisfaisable ($0 = 0$), le système obtenu **S'** est sous forme réduite et dans ce cas **S = S'**,
(3) **c'** est de la forme $x = r$ (x non astreinte à être non négative), le système **S'** est équivalent à $S \cup \{c\}$ et est sous forme réduite (sous réserve que toutes les contraintes de type $\neq$ liées à la variable x soient vérifiées),
(4) **c'** est de la forme $x^+ = r$ (toutes les varia-

bles de **c'** sont astreintes à être non négatives), le processus Simplex est activé, il permettra de dire si S ⋃ {c} est équivalent à un système S' sous forme réduite.

La simplification d'une expression arithmétique a fait l'objet d'une étude particulière afin d'en améliorer les performances, les différents exemples testés ayant montré que ce traitement intervenait un très grand nombre de fois au cours de la mise sous forme normale et se révélait d'un coût non négligeable.

*c. RÉDUCTION DU SYSTÈME $S^+$*

Le système $S^+$ est sous forme réduite dans la structure des variables astreintes à être non négatives s'il est de la forme :
$\{y_1 = b_1 + \Sigma_j a_{1j} x_j, \ldots, y_n = b_n + \Sigma_j a_{nj} x_j\}$
et s'il satisfait aux conditions suivantes :
(1) $S^+$ ne contient que des variables astreintes à être non négatives,
(2) pour chaque équation, $y_l$ et les $x_j$ sont des variables distinctes,
(3) les $y_l$ sont des variables distinctes entre elles et ne figurent dans aucun membre droit des équations de $S^+$,
(4) les $b_i$ sont des constantes arithmétiques non négatives,
(5) le système $S^+$ ne contient pas de variables figées (variable qui prend la même valeur pour l'ensemble des solutions de $S^+$).

La réduction du système $S^+ \cup \{e\}$ où $S^+$ est sous forme réduite et **e** une équation de la forme $b + \Sigma_i a_i x_i = 0$ (k constante numérique pouvant être nulle, $x_i$ variables astreintes à être non négatives) se déroule en deux phases, tout d'abord la mise de **e** sous une forme normale **e'** pour le système S+, puis l'application d'une suite de transformations au système $S^+ \cup \{e'\}$ pour obtenir un système $S^{+\prime}$, soit sous forme réduite dans la structure des variables astreintes à être non négatives, soit non soluble.

**aa)** Mettre **e** sous une forme normale pour $S^+$, c'est substituer chaque variable $x_i$ de **e** par l'expression $e_i$ si l'équation $x_i = e_i$ figure dans $S^+$ et simplifier l'expression obtenue ; deux cas de figure peuvent se présenter pour l'expression simplifiée $e_0$ :
(1) il existe au moins une variable y de $e_0$ qui ne figure pas dans l'ensemble constitué des variables des équations $S^+$, l'équation $y = b' + \Sigma_i a_i x_i$ est une forme normale de **e** pour $S^+$, elle est ajoutée à $S^+$,
(2) toutes les variables de $e_0$ figurent déjà dans l'ensemble constitué des variables des équations de $S^+$, on élimine d'abord de $S^+$ toutes les équations que l'on peut atteindre par une variable quelconque de $e_0$, puis on ajoute $e_0$ sous une forme normale au nouveau système $S^+$ (les équations éliminées, qui pourront toujours s'écrire sous forme normale, seront retraitées par la suite).

**bb)** Les transformations de $S^+ \cup \{e'\}$ sont effectuées au moyen de l'opération de pivot selon la méthode de Gauss-Jordan et doivent respecter les trois critères suivants :
(1) conserver les constantes de $S^+$ non négatives (elles peuvent varier) pour pouvoir exhiber une solution particulière,
(2) ne pas introduire la variable figurant en membre gauche de **e'** dans $S^+$ pour conserver la propriété : $S^+$ ne contient pas de variables figées,
(3) éliminer les cas de dégénérescences pour ne pas créer de cycles dans l'algorithme. Le choix du pivot permet de satisfaire ces trois critères.

Ces transformations s'arrêtent lorsque **e'** est de l'une des trois formes suivantes :
(1) $y = b + \Sigma_i a_i x_i$ avec b et les $a_i$ strictement négatifs, le système $S^{+\prime}$ n'admet pas de solutions,
(2) $y = b + \Sigma_i a_i x_i$ avec k strictement positif, le système $S^{+\prime}$ est sous forme réduite dans la structure des variables astreintes à être non négatives donc soluble,
(3) $y = \Sigma_i a_i x_i$ avec les $a_i$ strictement négatifs, le système $S^{+\prime}$ est soluble (aux contraintes de type = près) mais il n'est pas sous forme réduite dans la structure des variables astreintes à être non négatives puisqu'il contient au moins y et les $x_i$ comme variables figées à zéro.

*d. ÉLIMINATION DES VARIABLES FIGÉES*

Dans le cas particulier où la terminaison des transformations aboutit à la détection des variables figées, il faut éliminer du système $S^+$ toutes les équations que l'on peut atteindre par les variables de **e'**, puis retraiter successivement ces équations (à l'exception de **e'**) afin de vérifier si d'autres variables ne se figent pas, auquel cas on réitère ce processus. Ce traitement est indispensable pour les variables astreintes à être non négatives car l'information x = k (k constante non négative) permet de relancer la vérification des contraintes de type = liées à ces variables.

Essentiellement, deux méthodes ont été étudiées pour réaliser ce traitement :
- l'une qui consiste à supprimer de $S^+$ les équations à retraiter, ce qui est coûteux en temps (les structures utilisées pour le codage des éléments de $S^+$ sont complexes) et en place mémoire (de nombreuses modifications doivent alors être mémorisées dans la pile de backtracking), mais qui présente l'avantage d'alléger temporairement le système $S^+$,
- l'autre qui se borne à marquer dans $S^+$ les équations à retraiter de façon à ne pas en tenir compte lors de la recherche du pivot, mais qui présente le gros inconvénient d'effectuer les calculs sur les coefficients des équations qui n'ont pas encore été retraitées.

Ces deux méthodes ont jusqu'à maintenant donné des résultats comparables sur le plan des performances.

### 3. LE CODAGE DES ÉLÉMENTS DE $S^+$

Le choix des structures a été fortement guidé par celui déjà fait au niveau de l'interpréteur ; les différents éléments manipulés (variables et coefficients de $S^+$ principalement) sont donc représentés par une suite contiguë de mots machine dont le premier en détermine la nature.

Les systèmes traités étant généralement très creux (variables peu liées entre elles) et faisant intervenir un nombre important de variables, une représentation matricielle classique sous forme de tableau était peu réaliste. En outre, un accès très rapide aux éléments d'une "ligne" ainsi qu'à ceux d'une "colonne" s'avérait d'une importance non négligeable (recherche du pivot, élimination des variables figées et marquage des éléments pour la récupération de mémoire entre autres). C'est donc un codage mixte conciliant ces deux aspects qui a été adopté. Il s'agit d'un chaînage horizontal et vertical, avant et arrière, des coefficients du système (seuls les coefficients non nuls sont mémorisés), de plus, chaque coefficient possède un accès sur les variables dont il dépend. Ce codage permet de considérer le système sous sa forme duale ou primale. Une optimisation importante, liée aux structures, a été d'utiliser deux listes distinctes pour représenter un vecteur de coefficients, l'une contenant les coefficients positifs, l'autre les négatifs, ce qui a permis de minimiser les parcours, le signe des coefficients étant un élément déterminant dans l'algorithme de recherche du pivot.

## Les contraintes Booléennes

Pour en terminer avec ces nouvelles algèbres, PROLOG III sera donc capable de traiter des systèmes de contraintes booléennes. L'objet de ce chapitre est de présenter succinctement la représentation et les algorithmes de traitement de ce type de contraintes.

### 1. LA REPRÉSENTATION DES CONTRAINTES BOOLÉENNES

Ces contraintes seront représentées sous la forme d'équations booléennes du type : F S F' où F et F' sont des formules booléennes et S un symbole relationnel.

Dans cette représentation, les formules obéissent aux définitions classiques des wff (well formed formulae) du calcul propositionnel muni des opérateurs décrits ci-dessous :

- les constantes 1' (vrai) et 0' (faux),
- les opérateurs &, I, ¬ (respectivement et, ou, non),
- les symboles relationnels =, ≠, ⇒ (respectivement égal, différent et implique).

Par ailleurs, on utilisera des contraintes unaires du type : A : booléen.

### 2. LES ALGORITHMES DE RÉSOLUTION

Le traitement de systèmes de ces contraintes booléennes au niveau de l'unification nous a amené à étudier divers algorithmes de vérification de cohérence d'ensembles de formules en calcul propositionnel.

Pour être adapté à une utilisation répétée au sein même de l'interpréteur, l'algorithme choisi devait au moins remplir les conditions suivantes :

- Efficacité maximale en temps de calcul sur des ensembles importants de clauses.
- Nécessité d'être pourvu de fonctionnalités internes proches de celles de l'interpréteur.
- Adaptabilité aisée aux structures de données communes aux divers modules.

La solution que nous avons adoptée s'inspire très fortement des travaux réalisés récemment par Pierre Siegel dans le domaine du calcul propositionnel.

Il en résulte un algorithme utilisant à la fois la SL-Resolution (Kowalski-Küehner 1971) et des résultats concernant la production de clauses résolvantes incluses dans l'ensemble de clauses initial dans un but de simplification maximale du système courant.

Très succinctement cet algorithme s'appuie sur les résultats suivants :

Soit S un ensemble de clauses cohérent. Soit C une clause. Alors, en construisant une famille d'arbres de racine C, et en définissant une discrimination des littéraux de chaque clause de ces arbres en A-ancêtres, B-ancêtres, littéraux en production et littéraux effacés, on montre que :

(i) Si S ∪ C implique la clause vide alors on sait construire un arbre de racine C tel que tous les littéraux de C sont effacés.

(ii) Pour toute clause C' impliquée par S ∪ C, on sait construire un arbre de racine C tel que les littéraux en production sont exactement ceux de C'.

D'autre part, on sait que :

(iii) Soit S un système, C une clause de S et C' une clause contenue dans C alors les systèmes S et S ∪ {C'} - {C} sont équivalents.

Ces résultats nous permettent donc de ne produire effectivement que des clauses qui minimisent le système initial.

D'un point de vue développement, la mise en palce d'un tel algorithme, par un souci d'efficacité optimum, nous a conduits à l'étude de plusieurs points importants :

(i) L'élaboration de structures de données complexes permettant, d'une part, le plus possible d'accès directs sur les variables, les clauses et les littéraux qui interviennent dans l'arbre de résolution et, d'autre part, l'optimisation d'un grand nombre de primitives de calculs très souvent utilisées.

(ii) La recherche du plus grand nombre possible de méthodes ayant pour but de minimiser les parcours de ces familles d'arbres – par essence très coûteux – en éliminant certains points de choix inutiles.

Ceci est réalisé, notamment, en éliminant physiquement les littéraux effacés, et en bloquant au plus tôt les parcours amenant à la production de clauses dont on peut savoir, par avance, qu'elles seront peu intéressantes, généralement parce qu'elles ne sont incluses dans aucune clause du système ou bien du fait qu'une au moins des clauses produites les contienne au sens large.

### 3. PERSPECTIVES A COURT TERME

Les travaux que nous menons actuellement dans le cadre de cette étude portent plus particulièrement sur les points suivants :

- La mise en place du lien qui devra être fait avec les autres modules décrits dans les paragraphes précédents.
- La possibilité d'obtenir, à tout moment, un système simplifié, équivalent au système courant mais ne portant que sur un sous-ensemble des variables utilisées.
- L'étude de la possibilité d'utiliser la récupération de mémoire en cours de traitement des contraintes booléennes.

## Collaborations industrielles :

Participation au programme ESPRIT.

Le GIA est l'un des contractants du projet P1219 (1106) intitulé : "Further development of PROLOG and its validation by KBS in technical areas". Dans ce projet, le GIA a pour partenaires la société PrologIA (France) et les sociétés DAIMLER-BENTZ et Robert BOSCH (RFA).

Il s'agit de réaliser un système expert de diagnostic de pannes de voitures, écrit au moyen d'une version améliorée de PROLOG II. Dans ce cadre, le GIA doit ajouter à la version existante une arithmétique complète et l'intégration des contraintes numériques au mécanisme d'unification de PROLOG. DAIMLER-BENTZ et BOSCH utiliseront leur savoir-faire pour constituer la base de connaissance et les règles d'expertise dans le domaine choisi. C'est sur ces deux préalables que sera construit le système expert envisagé.

La société PROLOGIA commercialise différentes versions de Prolog sur plusieurs types de matériels : DEC, HP, SM90, IBMPC et compatibles, Apple II, MacIntosch...

# Nouvelles Architectures & Langages pour l'Intelligence Artificielle

**VLISP80** ***MOTS CLES***
aide à la documentation
apprentissage (classification de concepts)
architecture parallèle synchrone
architecture pyramidale
Common Lisp (langage de programmation)
compréhension automatique de programmes
environnements de programmation
graphes de partition
intelligence artificielle
méthodes de programmation
pour machines parallèles
normalisation des langages
prolog



## Références


[1] Wertz H., *"Intelligence Artificielle : application à l'analyse de programme"*, Masson, Paris, New York, Mexico, nov. 1984, 2e édition : mars 1987. Traduction angaise éditée par Ellis Horwood Limited, 1987. Traduction italienne éditée par Masson-Italia, 1986.

[2] Wertz H., *"LISP, une introduction à la programmation"*, Masson, Paris, New York, Mexico, juin 1985. Traduction espagnole éditée par Masson S.A. Barcelone, 1986. Traduction italienne éditée par Masson-Italia, 1986. Adaptation à CommonLISP et traduction anglaise éditée par John Wiley and Sons, Chichester, 1988.

[3] Wertz H., *"bVLISP: an Example of a Programming Environment"*, Proc. Hawaii International Conference on System Sciences 19, volume IIa, Software, jan. 1986, pp. 598-607.

[4] Wertz H., *"bVLISP: a Programming Environment for Exploratory Programming"*, Recueil des Conférences, Convention Informatique, avril 1987, pp. 168-170.

[5] Wertz H., *"bVLISP: An Example of a Programming Environment"*, in *Computers and Education*, (Robert Lawler Ed.), John Wiley, 1988.


Motivations de l'équipe :

- L'étude et la réalisation d'interprètes à haut degré de portabilité de langages d'Intelligence Artificielle : COMMON-LISP, SCHEME, PROLOG.
- La création de méthodes de programmation et de langages pour architectures parallèles synchrone à très grande échelle, machines planes en grille ou machines pyramidales.
- La réalisation d'environnements de programmation, de mise au point, de communication, et de documentation associés.
- La réalisation d'outils d'analyse et de compréhension automatique de programmes.

## Langages d'Intelligence Artificielle

P. Greussay a développé en 1986 une version portable du standard LISP industriel **COMMON-LISP** sous Unix. Écrite en langage C, ce système a été porté sur VAX, SUN, et SM-90. Greussay a également mis au point dans les mêmes conditions une *version d'étude* du langage PROLOG-2 de Marseille, qui en donne un modèle très précis d'implémentation. Notre version du lisp lexical **SCHEME** a été complètement réécrite, pour être compatible avec le standard défini dans *Revised (3) Report on the Algorithmic Language Scheme*. Les valeurs multiples et un important noyau d'outils de mise au point ont été ajoutés. Cette version figure sur la liste de distribution de SCHEME diffusée par H. Abelson au M.I.T., et a été demandée par des organismes très variés (e.g. CERN, Université d'Utah). Elle est à l'heure actuelle commercialisée par la société SODIMA.
Thierry Porcher a porté SCHEME sur une nouvelle architecture de type RISC : KIM développé par SODIMA.

## Programmation d'architectures parallèles synchrones

Ces études sont menées en collaboration avec la société NCR, avec laquelle Patrick Sinz a organisé un concours national de projets, qui a permis la diffusion de machines à bases de multi-processeurs **Gapp** dans les laboratoires nationaux.

Nous nous sommes consacrés à la mise au point d'un langage de programmation de cette architecture nommé *Gapl,* variante parallèle de SCHEME, à la construction d'un répertoire d'algorithmes de base, développé par Marie-Noelle Rozin sous le nom de *programmation plane,* et à l'adjonction d'une architecture de routage pour les communications non-locales.

Roger Tanguy, assisté de Didier Sagot et Roland Ballesta, ont construit en 1987 une machine en tranches AMD 29300, à 100 nanosecondes de temps de cycle, pour le contrôle d'une grille de 288 processeurs élémentaires *Gapp.*
Un autre ensemble d'études est mené en collaboration avec l'Institut d'Électronique Fondamentale d'Orsay (F. Devos), pour la réalisation de langages et de systèmes d'exploitation pour une architecture de machine parallèle pyramidale dédiée au traitement d'image.

### 1. ARCHITECTURE DE ROUTAGE POUR MACHINES SIMD

Jacqueline Signorini a réalisé une architecture de routage pour le chip multiprocesseur **Gapp** NCR 45CG72.
Le réseau de routage de type hypercube et constitué de 16 routeurs, confère à la machine de base **Gapp** SIMD bit-série deux capacités nouvelles : 1) la communication non-locale entre processeurs-éléments non voisins, 2) la transmission de messages dont la taille est fixée à 64 bits. Cette réalisation a donné lieu à publication à I.E.E.E. AI'88, à Hitachi City, Japon.
Signorini a défini et implanté un jeu de micro-instructions qui, ajusté aux 13 lignes de contrôle de la machine SIMD, permet la programmation des nœuds-routeurs du réseau. Le contrôleur multiplexe l'un et l'autre jeu de micro-instructions en fonction de la valeur d'entrée d'une broche supplémentaire du chip.
Des configurations partielles de la grille des processeurs-éléments (arbres, anneaux, graphes quelconques) sont désormais possibles grâce à la programmation par routeurs.
L'architecture **Gapp-Routeur** est simulée en langage C sous Unix. Les applications suivantes ont été micro-programmées : sommation d'arbres, H-tree, anneau de processeurs, réseau sémantique, pour illustrer différents aspects de la programmation parallèle d'éléments de traitement.

### 2. PROGRAMMATION D'ARCHITECTURES PYRAMIDALES

Jean Méhat développe la partie logicielle du projet de machine parallèle de structure pyramidale *Sphynx* construite à l'Institut d'Électronique Fondamentale d'Orsay.

Méhat travaille sur le contrôle et la programmation des machines pyramidales Multi-SIMD. Le contrôle pose un problème d'architecture à cause 1) du volume des flux d'instructions vers les couches de processeurs, de l'ordre du Giga-bits par seconde, 2) de la nécessité de les synchroniser entre eux, en raison de l'aspect MIMD entre les couches.
Il a développé un modèle de contrôle, basé sur la division des processus suivant la classe de mouvements de données auxquels ils

***Responsable scientifique***
Patrick Greussay

***Organismes***
Département d'Informatique
Université Paris 8-Vincennes
2, rue de la Liberté, 93526 Saint-Denis
&
L.I.T.P.
2, place Jussieu, 75221 Paris
☎ (1) 43362525

***Participants***
Harald Wertz
Daniel Goossens
Jean Méhat
Jacqueline Signorini
Roger Tanguy
Patrick Sinz
Marie-Noëlle Rozin
Thierry Porcher
Vincent Lesbros
Philippe Krief

prennent part (ascendant vers le sommet ou descendant vers la base) suffisamment puissant pour permettre des mouvements de données variés (e.g. croisements) et suffisamment simple pour donner un débit d'instruction tenant les processeurs occupés.

A cet effet, il a développé un langage de programmation pour les architectures pyramidales, **Pyr-e**, qui adresse les deux aspects SIMD et MIMD. Il est construit comme une extension du langage C.

L'aspect SIMD est traité par l'adjonction au C d'un domaine, *parallèle* ou *sériel,* qui spécifie si une variable est traitée dans l'ordinateur hôte ou dans les processeurs élémentaires d'une couche. Le domaine est spécifié au moment de la déclaration des variables et le domaine des expressions est déduit de celui des variables qui la composent. Si nécessaire, des conversions de domaines sont effectuées.

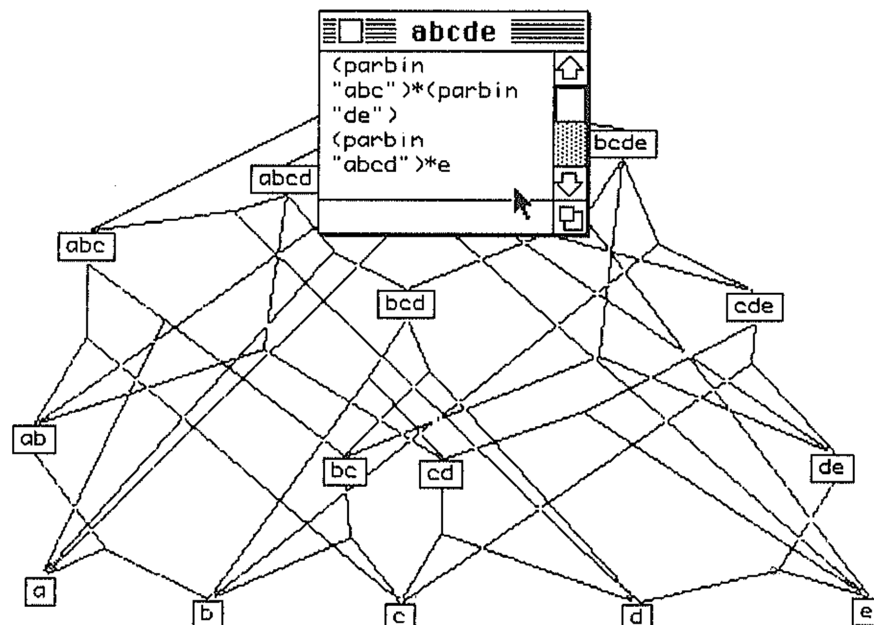
De même, le domaine des structures de contrôle est déduit de celui de leurs expressions de test. Les structures de contrôles du domaine parallèle sont traduites par des manipulations de l'ensemble des processeurs qui exécutent les instructions. Cette programmation, *au niveau des processeurs,* permet une grande souplesse de la programmation.

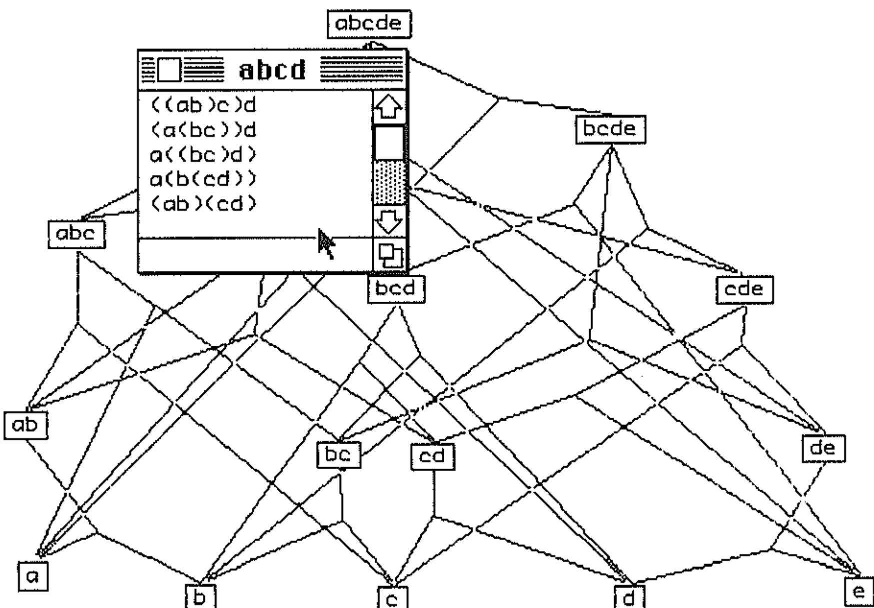
L'aspect MIMD est traité par les *transmissions de contrôle* : les activités dans une couche sont initialisées par celles des couches voisines. De cette façon, des processus locaux sont créés et disparaissent au fur et à mesure de la progression des mouvements de données. Les contrôleurs de couche permettent d'éviter les collisions entre mouvements de données.

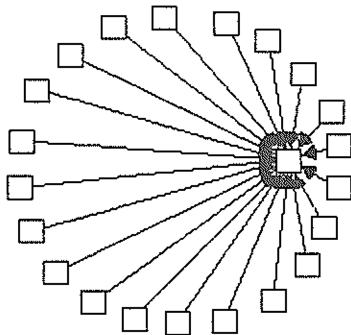
Le compilateur **Pyr-e** a été développé sous Unix; il produit du code pour la machine pyramidale *Sphynx* réalisée à l'I.E.F.

### 3. NOUVELLES PERSPECTIVES

L'équipe examine depuis peu les perspectives d'architectures ouvertes par l'approche connexionniste : La thèse de Christophe Lecerf soutenue en 1987, développe des méthodes d'auto-apprentissage et d'auto-organisation pour des réseaux d'automates neuronaux.
Jacqueline Signorini travaille à présent sur la construction d'un système de CAO pour la programmation d'automates cellulaires à très grande échelle.





## Classifications de concepts par graphes de partitions

La programmation parallèle pour l'intelligence artificielle et la représentation des connaissances suppose la maîtrise de la propagation de messages dans des topologies pouvant être arbitrairement complexes.

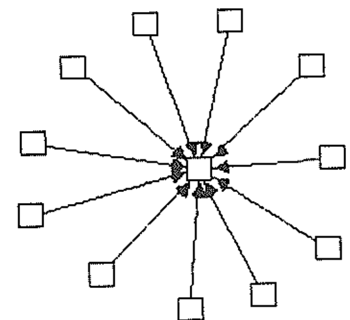
Daniel Goossens travaille actuellement sur l'intégration dans des systèmes de déduction automatique d'une opération de classification de concepts, basée sur une structure de données appelée *graphe de partitions.* Cette opération doit jouer un rôle complémentaire à l'opération d'*unification* en logique des prédicats. Elle prend en charge le processus essentiel et à long terme de comparaison de toute expression formelle nouvellement construite, avec les anciennes. La vérification d'une solution d'un exemplaire de cette opération est un problème co-NP complet.

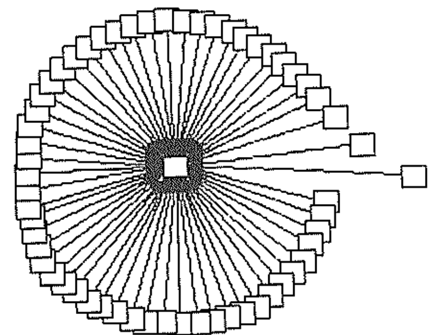
En juillet 1986, il a soumis à l'E.C.A.I. *(European Conference on Artificial Intelligence)* une première version complète de cette opération. Depuis, il a développé, implémenté et justifié théoriquement une série de stratégies de propagation de relations booléennes entre concepts dans les *graphes de partitions.*

Ces stratégies constituent des implémentations incomplètes de cette opération, et calculables en temps polynomial, ce qui en fait des outils de déduction fondamentaux utilisables en pratique.

Ces implémentations incomplètes s'inscrivent dans le cadre de l'étude systématique des graphes de partitions introduite par L.K. Schubert. Leur intégration dans les systèmes de déduction automatique est à mettre en parallèle avec les tentatives d'exploitation des connecteurs booléens dans les formules logiques, par exemple sous forme de règles de réécriture et d'unifieurs ACI comme il est fait chez J. Hsiang, et avec l'association de types structurés en treillis booléens, aux variables logiques, comme l'illustrent les travaux de A.G. Cohn. Un des buts de cette interprétation, la classification automatique d'expressions formelles, est illustré en analyse syntaxique du langage naturel par le classifieur de règles de grammaire du langage de représentation de connaissance KL-ONE réalisé par Schmolze et Lipkis. La base théorique provient en partie de l'explicitation par J.C. Reynolds de la structure de demi-treillis de l'ensemble des termes du premier ordre muni de l'opération d'unification.

La restriction des stratégies de propagation à des stratégies polynomiales participe à l'exploration systématique des stratégies poly-







[6] Greussay P., *"Comment programmer un multiprocesseur synchrone : connexion des données et propagation des signaux"*, Conférence invitée, Journées Afcet d'Études sur les Langages-Orientés-Objets, Paris, 5-7 janvier 1986, BIGRE-GLOBULE, n° 48, pp. 135-148.

[7] Greussay P., *"Machines à comprendre"*, in *Le texte en mouvement* (R. Laufer Ed.), P.U.V., 1987, pp. 11-21.

[8] Greussay P., *"LISP en France : 1971-1983"*, Colloque sur l'Histoire de l'Informatique en France, Grenoble, 3-5 mai 1988.

[9] Goossens D., *"Intelligence Artificielle et classification d'expressions formelles"*, Convention Informatique, 1987, tome B, pp. 176-181.

[10] Goossens D., *"Automatic Node Recognition in Graph"*, in *Advances in Artificial Intelligence II* (Du Boulay, Hogg, Steels Eds.), North Holland, 1987, pp. 327-333.

[11] Goossens D., *"La bibliothèque Édition-de-Graphes"*, Rapport Technique G-1, Université Paris-8, décembre 1987.

[12] Méhat J., et al., *"A Pyramidal System for Computer Vision"*, in *Pyramidal Systems for Computer Vision* (V. Cantoni, S. Levialdi Eds.), NATO ASI Series, Springer-Verlag, 1987.

[13] Signorini J., *"Un routeur pour le chip GAPP : étude, simulation et programmation"*, D.E.A. Intelligence Artificielle, Université Paris-8, octobre 1986.

nomiales de calcul de relations de subsumption entre expressions formelles en général, introduite par R.J. Brachman et H.J. Levesque.

Parallèlement, Goossens a implémenté en C sur MacIntosh une bibliothèque de primitives d'édition interactive de graphes à partir de laquelle il a entre autres implémenté un éditeur interactif de graphes de partitions.

## Environnements de programmation : bVLISP

L'étude des environnements de programmation est devenue un important domaine de recherche qui complète et prolonge la conception de langages : on peut mentionner l'ensemble des travaux effectués à Brown University, et ceux menés par N. Habermann à l'Université de Carnegie-Mellon.
Harald Wertz a conçu l'environnement de programmation **bVLISP** qui a permis de mettre au point une méthodologie d'implémentation fondée sur les concepts d'*attaches* et d'*assertions.*

Ce système a donné lieu à de nombreuses publications décrivant ses capacités d'édition, de documentation, et d'aide à la mise au point. Les travaux récents ont permis une clarification et une simplification des primitives d'*implémentation* de tels systèmes.
**bVLISP** a été développé initialement en VLISP, le système LISP à liaison dynamique conçu par P. Greussay. Ses diverses capacités d'aide à la lecture, à l'observation et à la modification de programmes LISP sont dues à deux modifications de l'interprète LISP :

1) une représentation de listes permettant l'attache d'informations supplémentaires aux champs usuels, 2) un nouveau noyau d'interprète *(eval, apply, liaison-déliaison* de paramètres) capable de prendre en compte la présence d'*attaches.*

Le module *eval* permet le lancement d'activités attachées à des points arbitraires des programmes à évaluer : évaluation d'activités d'observation ou de vérification associées à des fonctions standard.

Le module *apply* donne des capacités correspondantes pour les fonctions définies par l'utilisateur.

Enfin le module de *liaison-déliaison* permet l'évaluation d'activités d'observations ou de vérifications attachées à des variables.

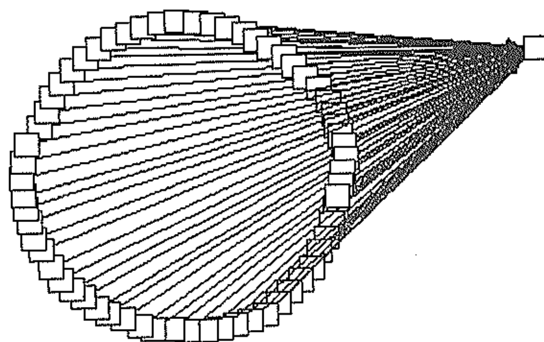
Dans les systèmes LISP à liaison dynamique, ces capacités étaient en germe dans les constructions telles que *evalhook* et applyhook. C'est lors d'une adaptation expérimentale de **bVLISP** à SCHEME que Wertz a mis au point une nouvelle construction : *bind-unbind-hook.* Il semble que ces trois constructions soient suffisantes pour le portage de la totalité de l'environnement bVLISP à un LISP à liaison lexicale tel que SCHEME ou COMMON-LISP.

La structure d'implémentation d'environnement de programmation ainsi dégagée peut se transposer à des langages très différents de LISP : l'équipe analyse actuellement les possibilités de modifications d'un interprète PROLOG pour y adjoindre les trois constructions de base. On peut en espérer rapidement une version de PROLOG comportant l'édition et la maintenance de *versions* de programmes, des outils de trace, d'observation et de vérification, ainsi qu'un module de documentation automatique et incrémentale de programmes PROLOG.

## Collaborations industrielles, contacts internationaux

Depuis un an, Harald Wertz, Vincent Lesbros et Philippe Krief collaborent avec la société Thompson pour la construction d'un système interactif et graphique de spécifications de projets.

Cette étude est soumise à la C.E.E. pour devenir un projet Esprit, en collaboration avec DELPHI (Italie), la Politechica de Milan (Italie), Xerox (France), l'université de Dortmund (RFA) et la société PCS (RFA). Dans ce projet les techniques d'évaluation symbolique et de représentations multiples de programmes mises au point par l'équipe seront intensivement utilisées.





[14] Signorini J., *"A Routing Model for the NCR GAPP"*, IEEE AI'88, International Workshop on Artificial Intelligence for Industrial Applications, Hitachi City, Japan, May 1988.

[15] Tanguy R., *"Un réseau d'automates programmables pour l'apprentissage de la communication"*, Thèse, Université Paris-6, décembre 1987.

[16] Lecerf C., *"Contribution à la modélisation des réseaux neuronaux et à l'apprentissage automatique dans les ordinateurs parallèles"*, Thèse, Université Paris-8, décembre 1987.

[17] Rozin M.-N., *"Étude pratique des ressources de la programmation plane"*, Rapport Interne, Université Paris-8, juillet 1987.

[18] Porcher T., *"Les implications du temps-réel dans l'architecture des machines symboliques. Exemple : les commutations de tâches dans KIM"*, Colloque AFCET Reconnaissance des Formes et Intelligence Artificielle, Antibes, novembre 1987.

# Mise en œuvre des langages de la logique

MALI *MOTS CLES*
Prolog
mise en œuvre
gestion de mémoire
non déterminisme
arbres infinis
interprète, compilateur
unification
temps réel
processeur spécialisé
coprocesseur
carte mémoire
IBM PC
VLSI

Le thème central du projet est la réalisation d'interprètes et de compilateurs de langages basés sur la logique des prédicats dont Prolog est le représentant le plus connu. Nous nous attachons à trouver une solution efficace aux problèmes spécifiques de gestion de mémoire posés par ces langages.
Notre objectif principal est de fournir aux concepteurs de tels langages des mémoires tant matérielles que logicielles, possédant une récupération automatique et qui soient aptes à être employées dans toute nouvelle mise en œuvre de compilateur ou d'interprète.
Concernant les caractéristiques des mises en œuvre que nous offrons, nous avons deux soucis principaux qui sont l'efficacité du point de vue mémoire et la largeur de leur spectre d'utilisation. De plus, nous respectons une contrainte importante de conception : dans les produits que nous développons, la récupération de mémoire doit pouvoir être effectuée en parallèle ou en semi-parallèle avec l'interprétation. Pour la récupération en parallèle, nous sommes évidemment amenés à développer du matériel spécifique. En particulier, c'est ce qui a été fait pour les microcalculateurs compatibles IBM PC pour lequel nous avons développé une carte mémoire comportant un coprocesseur microprogrammable incorporé sur la carte qui se charge de la récupération en parallèle de l'interprétation.

Notre activité est orientée dans trois directions :
- spécification d'une machine abstraite intermédiaire appelée MALI, qui offre des possibilités de création, modification, conservation d'objets structurés de haut niveau aptes à représenter l'état d'un interprète ;
- réalisation de diverses implantations de MALI ;
- réalisation d'interprètes Prolog utilisant MALI.

Les paragraphes qui suivent traduisent ces trois types d'activités.

## Conception de MALI

Afin de pouvoir supporter Prolog II, le répertoire de commandes de MALI a été entièrement redéfini ; il possède toutes les commandes de l'ancienne version, mais comporte des notions supplémentaires : unification de termes rationnels, variables à attribut (pour les coroutines de Prolog II), gestion interne à MALI du parcours de termes rationnels, création incrémentale de termes, possibilité de détruire un intervalle de points de choix au sein de la pile des choix (coupure de "tronçons").

Le fonctionnement interne de cette nouvelle version de MALI a été spécifiée en langage de haut niveau : cette spécification sert de base à la microprogrammation manuelle de la carte MALI pour IBM PC (voir paragraphe suivant).

## Implantation matérielle de MALI

### CARTE MALI POUR IBM PC

La carte MALI au format PC (voir rapport d'activité 86) a été mise au point ; elle peut être insérée aussi bien dans un PC-XT que dans un PC-AT, le choix se faisant par quelques "straps" sur la carte. Pour valider le bon fonctionnement de la carte, nous l'avons utilisée avec notre précédent système Prolog expérimental. Actuellement, la nouvelle version de microprogramme MALI est en cours de programmation d'après notre spécification en langage de haut niveau. Ce travail est effectué avec la collaboration d'un ingénieur de la société CRIL ; cette version devrait être au point en décembre 87.
Nous avons été amené à écrire un système de mise au point et de test du microcode : ce système de mise au point a été rédigé en Prolog II augmenté de quelques prédicats évaluables pour dialoguer avec la carte MALI installée sur le PC ; le Prolog II utilisé est notre propre version de Prolog II bâtie sur une version logicielle de MALI.
Enfin, nous commençons actuellement le transport de notre Prolog II sur la nouvelle version matérielle de MALI. L'interface de notre Prolog II avec MALI a été conçue de telle façon que son accrochage à la version matérielle de MALI soit relativement simple : le transport consiste en une redéfinition de macros C. Ce transport est en cours et devrait être terminé pour la fin de l'année 87 ; nous aurons alors un système Prolog II fonctionnant avec une version matérielle de MALI sur IBM PC (système MS-DOS), offrant une récupération de mémoire en temps réel.

### RÉALISATION D'UN CIRCUIT VLSI APTE A LA MICROPROGRAMMATION DE MALI

Grâce à une aide financière du CNET, nous avons pu continuer cette année la conception d'un circuit intégré CMOS pour la réalisation d'une nouvelle carte MALI. Deux circuits ont été dessinés au début de l'année. Leur cuisson a été confiée au CMP. Le CNET de Grenoble a reçu le premier (CMOS un métal) qui rassemble divers motifs de test. La société MHS s'est occupée du deuxième circuit (CMOS double métallisation) qui implémente une mémoire à double accès avec arbitre intégré. Le test fonctionnel du premier circuit, déjà réceptionné, reste à faire. Le deuxième circuit n'est pas encore arrivé.



## Références


**Thèses**

[Ridoux 86] O. Ridoux, *"La gestion de mémoire temps réel des langages de programmation relationnels"*, Thèse 3e cycle, Université de Rennes I, 5 décembre 1986.

[Amraoui 88] M. Amraoui, *"Une expérience de compilation de Prolog II sur MALI"*, Thèse 3e cycle, Université de Rennes I, 23 janvier 1988.

**Communications à congrès et colloques**

[Chevallier 87] L. Chevallier, S. Le Huitouze, O. Ridoux, *"Style de programmation pour une machine de programmation logique munie d'un récupérateur de mémoire"*, Séminaire 1987 de programmation logique, Trégastel, 19-21 mai 1987.

[Ridoux 87] O. Ridoux, *"Deterministic and Stochastic Modeling of Parallel Garbage Collection - Towards Real-Time Criteria"*, 14th Symposium on Computer Architecture, Pittsburgh, Pennsylvanie, 2-5 juin 1987.

***Responsable scientifique***
Y. Bekkers

***Organisme***
INRIA - Rocquencourt
Domaine de Voluceau
Rocquencourt
78150 Le Chesnay
☎ (1) 39635511

Université de Rennes

***Participants***
B. Canet
O. Ridoux
L. Ungaro
L. Chevallier
S. Le Huitouze
M. Amraoui
H. Lacourt

Parallèlement à cette activité de réalisation de bas niveau dont le but est la familiarisation avec la technologie des circuits intégrés CMOS, la conception du circuit se poursuit. La prise en considération de la position particulière occupée par le circuit intégré dans l'architecture globale :
- coprocesseur au service de l'hôte,
- processeur autonome pour son activité de gestion mémoire,

et l'importance des interfaces pour ce circuit (avec l'hôte, avec la mémoire des termes) nous ont orientés vers certaines solutions concernant le cadencement du circuit et son architecture.

La définition des unités fonctionnelles constituant la machine MALI et le dessin de la frontière qui isle le circuit intégré du reste de la carte ont été effectués. Le brochage du circuit s'en est trouvé ébauché, il reste à le préciser : un bus de 32 bits avec adresses et données multiplexées permet d'accéder à la mémoire des termes et à la mémoire commune, un bus de 16 bits lui aussi multiplexé relie le circuit à sa mémoire de microcode. L'architecture interne du circuit au niveau registre est en train d'être spécifiée à l'aide du simulateur logique HILO (système de simulation de circuits logiques) ; le jeu d'instruction du processeur dont le point de départ est celui de la maquette déjà existante est lui aussi en chantier.
Une stratégie de cadencement particulière dite "autosynchronisée", par opposition au cadencement synchrone, a été choisie. Un simulateur adapté à la modélisation de ce genre de machine a été écrit pour valider les interconnections intra-modules du circuit.

## Implantation logicielle de MALI en "C"

L'implantation logicielle de MALI réalisée sur IBM PC l'année passée a été transportée cette année sur machines UNIX, en particulier sur SUN et HP 9000 UNIX. Un autre transport a été effectué sur VAX 750 et a été intégré au logiciel Prolog/P.
L'effort sur cette version logicielle de MALI a aussi porté sur sa documentation afin d'en faciliter sa diffusion. Un document complet est en cours de rédaction.
Les performances en vitesse sur IBM PC de cette version logicielle de MALI nous ont agréablement surpris puisque cette version de MALI a permis la mise en œuvre d'un interprète Prolog II deux à trois fois plus rapide sur la même machine que l'interprète Prolog II original commercialisé par la société Prolog IA.

## Réalisation d'interprètes Prolog

### RÉALISATION D'UN INTERPRÈTE PROLOG II

L'interprète Prolog II/MALI, commencé l'année dernière, a été enrichi par l'adjonction de nombreux prédicats évaluables. Le logiciel est maintenant entièrement compatible avec le Prolog II de Prolog IA. L'indexation des clauses est une technique généralement réservée aux compilateurs Prolog à la seule fin d'améliorer les performances en temps d'exécution. Nous l'avons rajoutée à notre interprète pour deux raisons. Premièrement, l'unification de certaines têtes de clauses est purement et simplement supprimée car on détecte qu'elle échouerait de toute façon. Ceci a un effet direct sur la vitesse d'exécution. La deuxième raison est une raison plus logique. Le fait de supprimer certaines unifications diminue le non-déterminisme et donc le nombre de points de choix sauvegardés lors d'une résolution. Outre le gain de temps provoqué par l'absence de commandes de sauvegarde, le volume d'informations dynamiques nécessaires est réduit, ce qui entraîne une meilleure utilisation de la mémoire de MALI.
Les mesures effectuées (voir plus haut) ont montré que Prolog II/MALI se comporte bien.

Actuellement, deux transports de cette version sont en cours :
- une version sur IBM PC utilisant la carte MALI,
- une version sur SUN.

### RÉALISATION D'UN INTERPRÈTE PROLOG/P

Un ingénieur de la CRIL (voir actions industrielles) a été accueilli à l'IRISA dans le but de le former aux techniques de mise en œuvre des langages de programmation logique, et particulièrement aux principes de gestion de mémoire conçus dans l'équipe (la machine abstraite MALI), et de diffuser ces techniques auprès d'un éventuel partenaire industriel.
Une partie de cette formation a consisté en la réalisation d'un système Prolog compatible avec celui que la CRIL distribue (Prolog/P). Pour ce transport, une partie du système Prolog/P a été récupérée. En effet, nous avons conservé l'analyseur syntaxique, le générateur de la représentation interne des programmes et les interfaces utilisateur et système du produit final écrit en Pascal. Nous y avons greffé un nouvel interprète, de nouvelles procédures d'évaluation des prédicats évaluables et bien entendu la mémoire MALI elle-même, le tout est écrit en C.

## Étude des problèmes de compilation en relation avec MALI

### COMPILATION DE PROLOG

Cette année, nous avons démarré l'étude de la compilation de Prolog et l'influence sur celle-ci des techniques de gestion de mémoire. Nous avons considéré deux techniques : la technique procédurale de Warren, qui induit la machine abstraite de Warren (WAM), et la machine MALI, qui spécifie sa propre gestion de mémoire et sert de machine cible à la compilation. Les deux manières de compiler sont très différentes pour tout ce qui met en jeu la gestion de mémoire. Toutefois, une approche descendante met en évidence un niveau de détail où les deux manières sont indifférenciées. En ce qui concerne la généralisation des deux manières de compiler à des stratégies d'évaluations nouvelles, on constate à nouveau d'importantes différences.
La machine WAM, parce qu'elle est fondée sur une technique de gestion de mémoire procédurale où la pile des appels est l'élément principal, ne peut pas servir à l'interprétation efficace d'un langage de programmation logique où la stratégie d'évaluation ne serait plus séquentielle (ex. de gauche à droite) mais concurrente. La machine MALI étant indépendante de la stratégie d'évaluation conserve ses capacités quelle que soit la stratégie.
Actuellement, nous nous engageons au côté de la société CRIL dans une phase de développement d'un compilateur industriel utilisant MALI (voir paragraphe actions industrielles).

### UNE EXPÉRIENCE DE COMPILATION DE PROLOG II SUR MALI

Nous avons réalisé un compilateur expérimental de Prolog II, qui utilise la version logicielle de MALI. Ce compilateur est basé sur la définition d'un ensemble de pseudo-instructions d'une machine abstraite. L'implémentation consiste à coder ces pseudo-instructions par des commandes MALI. Le code généré par le compilateur est un programme "C" qu'il faut recompiler à son tour puis lier au système MALI. Le compilateur est écrit en Prolog.
Cette réalisation a été menée dans une perspective d'intégration avec l'interprète Prolog II existant, ce qui a nécessité l'utilisation des mêmes structures de données et des mêmes stratégies de réalisation des prédicats prédéfinis tels que le "Dif" et le "Geler", etc.

Cette réalisation a été conçue en dehors de tout souci d'efficacité dans un but didactique afin de mieux mesurer les problèmes d'intégration d'un compilateur et de la machine MALI. Une thèse de M. AMRAOUI présente le bilan de cette expérience.

## Actions industrielles

Actuellement, notre équipe a des relations privilégiées avec le centre de Rennes de la société CRIL. Cette année, nous avons pu former un ingénieur de cette société aux techniques de gestion de mémoire dans les interprètes Prolog, en particulier à l'utilisation de MALI. En effet, grâce à un contrat tripartite entre cette société, l'IRISA et l'EPR, un ingénieur de cette société a pu passer six mois dans notre équipe. Durant les quatre premiers mois, nous avons effectué en commun le transport de l'interprète Prolog/P de la CRIL sur MALI. Les deux derniers mois ont été consacrés à une préétude des problèmes de compilation de Prolog.

Ce travail débouche maintenant vers des perspectives industrielles pour notre équipe puisque la société CRIL s'engage actuellement dans le développement d'un compilateur Prolog utilisant MALI. Nous comptons participer activement à ce développement en fournissant une mémoire MALI logicielle adaptée au compilateur de la CRIL.

Un deuxième ingénieur de la CRIL, récemment intégré dans notre équipe, est en train de se familiariser avec nos réalisations matérielles. Il participe à la microprogrammation de la carte MALI pour IBM PC. Notre objectif pour l'année 1988 est de trouver un financement pour pouvoir entreprendre, après une période de mise au courant, une mise à niveau industrielle de notre carte compatible PC en vue de sa commercialisation.

# Implémentation et construction d'outils Prolog

**Responsable scientifique**
Marc Gillet

**Organisme**
IBM / Centre Scientifique
36, avenue Raymond-Poincarré
75016 Paris
☎ (1) 45 05 14 00

**Participants**
Jacques Bellone
Marc Gillet
Xavier de Lamberterie
Remy Picca
Gilbert Rouquié
Zeina Zein

## Présentation du thème

En 1987, le projet "Implémentation et construction d'outils Prolog" du Centre Scientifique IBM comprenait trois activités centrées sur Prolog :

- L'interprétation et la compilation de Prolog (VM/Prolog, Bayou).
- L'utilisation de langages objets pour la représentation des connaissances en Prolog (Glances).
- L'interprétation et la compilation du Concurrent Prolog.

## Principaux résultats obtenus en 87

### BAYOU

(Marc Gillet, Xavier de Lamberterie, Gilbert Rouquié, Zeina Zein)
Après la réalisation de VM/Prolog qui était une implémentation de Prolog de :
1. type "structure sharing",
2. avec un espace adressable de 24 bits,
3. composé d'un interpréteur et d'un compilateur incrémental.

Une nouvelle implémentation de Prolog appelée Bayou a été entreprise.

Bayou possède les caractéristiques suivantes :
1. type "structure copying",
2. avec un espace adressable de 31 bits,
3. composé d'un interpréteur et d'un compilateur incrémental et d'un compilateur complet,
4. possibilité de partager un espace de travail Prolog en mémoire centrale entre plusieurs utilisateurs.

A cette date :
- l'interpréteur et le compilateur incrémental sont terminés et testés,
- les prédicats prédéfinis sont terminés et en cours de test,
- une interface entre Bayou et les langages : assembleur, Fortran, Pascal, PL1 a été réalisée,
- le générateur de code du compilateur complet est terminé.

### GLANCES

(Zeina Zein)
Glances est un langage de représentation des connaissances implémenté en VM/Prolog et basé sur les concepts de la Programmation Orientée Objet.

En plus des caractéristiques classiques des langages orientés objet, Glances offre la possibilité de représenter des objets satisfaisant des contraintes. Deux modes de déclenchement des contraintes sont possibles : automatique et à la demande de l'utilisateur.
Glances est utilisé comme langage de représentation de la base de connaissances d'un système d'"Aide à la Sélection de produits bancaires" dans le cadre d'une étude menée conjointement avec une banque.
Cette année, le travail a porté sur l'achèvement de cette étude : construction d'une ***base de connaissances en grandeur réelle.***
L'approche ***Prolog-Objets*** s'est révélée bien adaptée pour élaborer et gérer des ***bases de connaissances importantes.***
Les performances actuelles du système sont satisfaisantes. Elles sont supérieures à celles de nombreux systèmes de schémas, en raison de la ***modularité*** de la base qui rend les accès très efficaces.
De plus, cette architecture permet l'***extension et la maintenance de la base,*** tout en gardant sa ***cohérence*** grâce aux ***mécanismes de contrôle.***

### IMPLÉMENTATION DE CONCURRENT PROLOG

(Jacques Bellone, Remy Picca)
Il s'agissait d'effectuer une implémentation de Concurrent Prolog dans le cadre d'une thèse de troisième cycle.
Cette thèse a été soutenue en septembre 87 et cette soutenance a terminé l'activité du projet sur Concurrent Prolog.

## Perspectives futures

### BAYOU

Le compilateur complet sera terminé en juin 88 et le restant de l'année sera consacré :
- au test du compilateur complet,
- à terminer le ramasse-miettes et à le tester,
- à réaliser le partage d'espace de travail en mémoire entre utilisateurs.

### GLANCES

L'année 1988 verra le développement d'un moteur d'inférence spécifique et d'un debugger.

# Programmation en Scheme et dans les Lisp lexicaux

Bordeaux

***Responsable scientifique***
Pierre Casteran
Cette nouvelle équipe et son projet viennent d'être accueillis tout récemment au sein du Gréco programmation au moment de la préparation de cet ouvrage.

***Organisme***
Université de Bordeaux I
UER de Mathématiques et Informatique
351, cours de la Libération
33405 Talence Cedex
☎ 56846089 & 56846090

***Participants***
Jean-Pierre Braquelaire
Myriam de Sainte-Catherine
Jean-Claude Royer
Robert Strandh

Plusieurs étudiants en DEA et en thèse seront également associés à ce projet.

LEXICO *MOTS CLES*
Lisp
programmation logique
programmation par objets
Scheme

## Construction d'un Environnement de Programmation

Nous nous proposons de construire un environnement autour d'un compilateur utilisant la conversion en code "CPS" (Continuation Passing Style) [STEELE, KRANZ et al.]. Cette technique consiste en la création d'un code intermédiaire, toujours en Scheme ; ce code, bien que peu lisible pour un lecteur humain, contient toutes les informations nécessaires à l'optimisation du code compilé : béta-réduction, analyse de fermetures, etc. Notre environnement doit pouvoir fournir en clair ces informations, afin de permettre une analyse de la qualité du code généré. Ces outils nous paraissent nécessaires, le langage Scheme encourageant plus que tout autre l'emploi intensif d'abstractions et de récursivité.
Une extension intéressante pourrait être la généralisation de ces outils à des schémas de programme, afin de raisonner sur des classes de programmes plutôt que sur des exemples particuliers.

## Applications

### PROGRAMMATION PAR OBJETS

Scheme a été choisi pour être le langage de commande du projet LUMIÈRE du Laboratoire d'Informatique de l'Université de Bordeaux I. Ce projet consiste en la réalisation d'un logiciel de synthèse et de manipulation d'images réalistes en deux dimensions.
Une adaptation d'OBJVLISP a été écrite en PCScheme, permettant notamment l'usage d'objets décrits sous plusieurs aspects.
La présence de ces multiples aspects soulève de nombreux problèmes :
- équivalence d'aspects,
- choix d'une représentation réelle ou virtuelle,
- héritages.

Dans le cadre plus précis de la synthèse d'images 2D, des expériences concluantes ont porté sur la structuration des couleurs et des régions, et la spécification de dégradés.

### PROGRAMMATION LOGIQUE

Nous nous proposons d'étudier deux modèles d'intégration avec Prolog :
- les continuations logiques [Haines],
- les continuations de succès [utilisées notamment dans le Prolog Symbolics].

Nous pensons que le choix d'un noyau basé sur les continuations doit faciliter l'étude de ces communications Lisp-Prolog.

## État actuel de ces Recherches

a) Réalisation d'une première maquette d'interprète Scheme en LE-LISP, par P. Castéran. Elle comprend la conversion en code CPS, et un interprète spécialisé dans ce code. Ce logiciel est utilisé dans les cours de troisième cycle sur Scheme (DEA et DESS).
b) Début (janvier 1988) du transport de ce programme sur station Symbolics 3620, l'utilisation des nombreux outils de cette machine (Flavors, outils de représentation de graphes) facilite la lecture du code CPS.
c) ObjScheme, l'adaptation d'Objvlisp en PCScheme, écrite par J.-C. Royer, a servi de base à l'écriture d'ITTEN, un logiciel pour le traitement informatique de la couleur. Une communication (P. Castéran, J.-C. Royer) a été soumise à EUROGRAPHICS 1988.
d) Une maquette de logiciel d'enseignement de la musique est en cours de réalisation en ObjScheme, sous la direction de Myriam de Sainte-Catherine. Elle s'appuie, entre autres, sur un noyau Prolog/Scheme développé par Patrice Lacoste et Jean-Luc Gibot-Leclerc, étudiants en Thèse du C.N.A.M.

## Relations avec d'autres équipes

*VLISP80* (Patrick Greussay).
Certains des outils développés seront intégrés à la version de Scheme écrite en C par Patrick Greussay (sur VAX/780, Silicon-Graphics Iris, Stations Sun).
*METHEOL* (Michel Billaud, G. Filé, M. Corsini).
La recherche sur l'intégration Scheme-Prolog se fera en collaboration avec cette équipe, notamment sur les problèmes de paramétrisation de la stratégie de recherche et la représentation de la liste des succès d'un appel Prolog par un flux infini (thèse de Michel Billaud).

## Moyens existants

Serveur GeoCub du Greco Programmation à Bordeaux, stations SUN, Silicon Graphics IRIS, et Symbolics 3620.

# Calcul symbolique et nouvelles architectures

orsay

***Responsable scientifique***
Ion Filotti

***Organisme***
L.R.I. Université d'Orsay
université de Paris Sud
91405 Orsay
☎ (1) 69 41 66 29 & 69 41 66 28

***Participants***
Patrick Amar
Jean-Paul Bodeveix
Eric Durocher
Marc Durocher
Ion Filotti
Karim Hebbar
Wladimir Mercouroff

## Présentation générale

Nous nous intéressons principalement à l'architecture logicielle et matérielle des langages fonctionnels et des langages pour la programmation logique.
De nouveaux modes de programmation se rajoutent depuis quelques années à la programmation fonctionnelle, outil de choix de l'intelligence artificielle, mais aussi et surtout modèle fondamental du calcul. Parmi ceux-ci, la programmation logique est la plus connue.
Les travaux peuvent être divisés comme suit :
- Lisp et programmation fonctionnelle.
- Systèmes d'exploitation.
- Programmation logique.
- Outils de programmation.

## Lisp et programmation fonctionnelle

Nous travaillons depuis plusieurs années sur les implantations efficaces de Lisp.

Patrick Amar a réalisé LAL, un interprète LISP très efficace écrit en C et donc parfaitement portable. Ce système a des performances supérieures à celles de LeLisp. Cet interprête est devenu récemment une implantation parfaitement compatible avec Franz Lisp. Elle est utilisée par l'équipe de programmation du L.R.I. dans la réalisation du système de spécification de types abstraits ASPEGIQUE. LAL est également utilisé par l'éditeur WINNIE pour son mécanisme d'extension.

Les travaux sur le coprocesseur microprogrammable KOALA sont terminés. Karim Hebbar a effectué les tests de Gabriel qui confirment nos thèses initiales sur l'intérêt de tels coprocesseurs. Nous disposons donc d'une machine Lisp à faible coût, réalisée intégralement au L.R.I.

La machine KOALA a été présentée à plusieurs manifestations importantes, dont la plus récente est celle organisée par la D.R.E.T. sur les machines symboliques en Décembre 1987.

## Systèmes d'exploitation

Les travaux sur une nouvelle génération de système d'exploitation sur des multiprocesseurs ont été poursuivis. Une première version d'un noyau modifié d'Unix BSD 4.2, avec processus légers a été réalisée par Eric Durocher sur la machine PANDA multiprocesseurs construite par l'équipe.

## Programmation logique

Jean-Paul Bodeveix continue ses travaux de réalisation d'un compilateur Prolog parallèle sur une structure multiprocesseurs à base de Transputers. Une première version du compilateur a été terminée sur un réseau de machines SUN. Il existe deux variantes du compilateur : l'une utilisant des processus synchronisés par messages, l'autre à parallélisme ET et variables partagées.

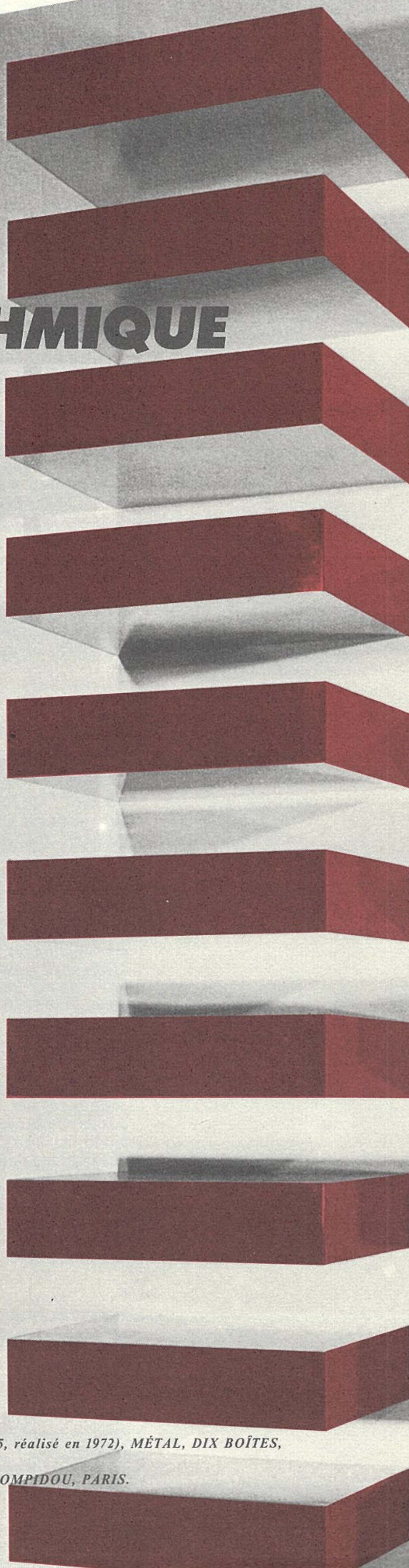
## Outils de programmation

Patrick Amar a réalisé XW, un nouvel éditeur de fichiers pour postes de travail graphiques, utilisant le système de fenêtrage X-Window.

L'éditeur fonctionne en mode serveur. Les processus clients ouvrent de véritables fenêtres du système x, dans lesquelles ils ont la possibilité d'éditer soit en utilisant les touches de l'éditeur vi, soit la souris et des menus.
Le fonctionnement en mode graphique et en mode serveur offre une meilleure ergonomie.



# LGORITHMIQUE



DONALD JUDD
*"STACK", 1972 (projeté en 1965, réalisé en 1972), MÉTAL, DIX BOÎTES, 24 x 103 x 80 cm chacune.*
*MNAM, CENTRE GEORGES-POMPIDOU, PARIS.*

## NOTE DE PRESENTATION

Philippe Flajolet

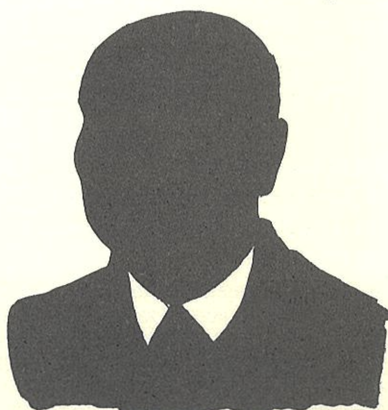


L'algorithmique est une discipline ancienne dont on peut faire remonter les premiers développements à l'Antiquité. Elle s'est longtemps intéressée aux calculs sur les nombres entiers et réels, et aux constructions de figures géométriques. Depuis l'avènement des ordinateurs, et leur très large diffusion, d'autres concepts ont fait leur apparition comme objets possibles d'un algorithme ; il s'agit par exemple des fichiers, des textes, des réseaux ou de schémas divers tirés de problèmes concrets. La discipline algorithmique s'est alors développée suivant deux axes : le premier consiste en la mise en évidence de structures de base fondamentales à l'aide desquelles se décrivent des algorithmes élaborés pour des domaines les plus divers ; le second consiste en la définition quantitative de la complexité d'un algorithme et la classification des algorithmes suivant leur complexité.

Depuis la création du GRECO Programmation, les projets en Algorithmique se sont développés en s'inscrivant dans le cadre de ces problématiques. De nombreux travaux ont montré que les notions de mots, d'arbres et de graphes jouent le rôle fondamental d'éléments de base dans le jeu de construction constitué par la conception d'algorithmes. Dans ce cadre, au sein du projet Algomots, a été développé un logiciel mettant en œuvre des algorithmes sophistiqués sur les automates et les mots. Ce logiciel intéresse à la fois l'informatique théorique et de nombreux autres domaines comme le traitement de textes en linguistique, le développement de nouveaux langages de programmation logique, ou d'autres domaines d'applications utilisant les automates. Dans la même voie, les algorithmes sur les graphes et leur dessin sur écran ont fait l'objet du projet LEG au sein duquel a été développé un langage de manipulation d'ensembles et de graphes.

Les axes de développement les plus récents de l'algorithmique montrent son universalité et son rôle de discipline fondamentale servant d'outil aux autres branches de l'informatique. Il en est ainsi pour les questions soulevées par des problèmes issus des bases de données, des protocoles de communication et même de certains aspects de l'intelligence artificielle.

Une place à part doit être donnée à tout ce qui concerne l'algorithmique géométrique. Son développement récent est spectaculaire ; il est lié à la diffusion du matériel graphique : le dessin de figures géométriques sur écran conduit tout naturellement à se poser des questions nouvelles. Toute une partie du projet Algo à l'École Normale Supérieure est orientée dans cette problématique et de nombreux nouveaux projets sont en cours de gestation. *Robert Cori*



# Analyse d'Algorithmes

**_Responsable scientifique_**
J.-M. Steyaert

**_Organisme_**
Centre de Mathématiques
de l'École Polytechnique
Plateau de Palaiseau
91128 Palaiseau Cedex
☎ (1) 60194091

**_Participants_**
P. Hennequin
B. Salvy
J.-M. Steyaert
École Polytechnique
Centre de mathématiques appliquées

ALGO *MOTS CLES*
algorithmes de tri
analyse d'algorithmes
calcul formel
manipulation symbolique d'expressions
recherche arborescente

Ce thème consiste en l'étude combinatoire des algorithmes de base de la programmation. Un phénomène courant est qu'un programme ne se comporte pas de façon uniforme sur les données selon leur complexité combinatoire ; cette problématique, largement développée par D.E. Knuth, est à l'origine de nombreux travaux sur l'optimisation des logiciels et des systèmes informatiques.

L'équipe travaille sur deux directions principales : les algorithmes de manipulation symbolique d'expressions logiques et algébriques et les algorithmes de tri et de recherche arborescente. Le premier thème est lié à la conception des logiciels de calcul formel et de déduction logique, le second est un des classiques de l'analyse d'algorithmes. L'objectif est de développer des méthodes systématiques d'analyse qui permettent de prédire quasiment "à vue" ce qu'on peut attendre du comportement moyen d'un programme. On s'attache donc à définir des méthodes de traduction des programmes en équations sur des séries qui rendent compte mathématiquement du coût des programmes, puis on utilise des propriétés analytiques des solutions pour obtenir les comportements asymptotiques cherchés.

P. Hennequin (boursier de thèse) a ainsi pu reprendre et étendre les résultats de R. Sedgewick sur l'analyse de "Quick-Sort". Il obtient une méthode de calcul des moments de la distribution des coûts pour un grand nombre de variantes de l'algorithme. Les résultats obtenus grâce à l'utilisation intensive du calcul formel permettent d'espérer une caractérisation de la distribution limite. Il travaille également sur des extensions au cas des clés répétées en collaboration avec J. Olivos (Université de Santiago du Chili).

J.-M. Steyaert, tout en dirigeant les travaux de P. Hennequin, travaille sur les algorithmes de simplification d'expressions en collaboration avec R. Casas (U.P.C. Barcelone) et M.-I. Fernandez Camacho (U. Complutense, Madrid), dont il assure de fait la direction de thèse. On montre que, sous de très larges hypothèses, le coût moyen est linéaire alors qu'il est en O(n.log n) dans le pire des cas. Comme ces algorithmes sont en général très simples, ceci tend à montrer que des algorithmes complexes et sophistiqués qui assurent la linéarité dans tous les cas sont moins efficaces en pratique.

La nature des calculs et des développements symboliques à effectuer se complexifiant très rapidement, il semble aujourd'hui fondamental de développer un système de calcul formel formel qui permette de manipuler effectivement les expressions mathématiques énormes qui sont produites par nos méthodes. Ce système devrait être mis en œuvre dans le courant de l'année et contribuer à l'étude de programmes encore plus complexes. C'est Bruno Salvy (X84), boursier de recherche, qui est chargé de développer la partie "Analyse asymptotique" de ce système en collaboration avec Paul Zimmerman (X84) qui travaille à l'INRIA sous la direction de P. Flajolet. B. Salvy a déjà réalisé une maquette de son système ; il faut en fait compiler de nombreux résultats parfois très fins issus de l'Analyse Complexe, et les intégrer dans un système de calcul formel. Le système choisi est Maple (bientôt diffusé par l'INRIA) en raison de sa relative transparence, de sa taille réduite et de son portage aisé sur de nombreuses machines Unix.

L'équipe est associée au sein du groupe "Algorithmes" au PRC Mathématiques-Informatique et à l'INRIA. Deux coopérations internationales (UPC Barcelone et TUW Vienne) sont coordonnées par J.-M. Steyaert.



## Références

J.-M. Steyaert et R. Casas, *"Bottom-up recursion in trees"*, Proceedings of CAAP (Nice, 1986, Lectures Notes in Computer Science n° 214, Springer Verlag.

P. Flajolet et J.-M. Steyaert, *"A Complexity Calculus for Recursive Tree Algorithms"*, Math. Systems Theory 19, 301-331 (1987).

P. Hennequin, *"Combinatorial Analysis of Quicksort Algorithm"*, à paraître dans RAIRO.

B. Salvy, *"Détermination automatique du comportement asymptotique des coefficients de fonctions génératrices"*, Rapport d'Option Scientifique, École Polytechnique.

## Séminaires

J.-M. Steyaert : Amiens (mai 1986), Journées du PRC Math-Info (février 1986), Journées Codage et Complexité (CIRM, mars 1986).

UPC Barcelone (mai 1986). TUW Vienne (décembre 1986).

Journées Groplan du Greco de Programmation (novembre 1987).

Universidad Complutense et Universidad Politechnica de Madrid (mai 1987).

# PANDORE : Un système expert pour l'optimisation des systèmes dynamiques

***Responsable scientifique***
J.-P. Quadrat

***Organisme***
INRIA
Domaine de Voluceau - Rocquencourt
BP 105
78153 Le Chesnay Cedex
☎ (1) 39 63 55 11

***Participants***
J.-P. Chancelier
C. Gomez
J.-P. Quadrat
A. Sulem

PANDORE *MOTS CLES*
calcul formel
FORTRAN
génération de programmes
génération de rapports
Macsyma (outil symbolique pour les mathématiques)
optimisation du calcul scientifique
optimisation de code
systèmes experts

L'objet du projet est de faire une tentative d'automatisation de l'ensemble des activités à accomplir dans une étude d'optimisation de systèmes dynamiques. Traditionnellement, seule l'activité purement numérique l'était.

Il existe de bonnes bibliothèques de programmes numériques pour l'optimisation dans $R^m$, l'intégration d'équations différentielles ou aux dérivées partielles, la résolution de systèmes linéaires. Ces programmes commencent à être intégrés dans des systèmes de taille importante plus ou moins conviviaux. Ils sont en grande majorité écrits en Fortran. Plus récemment, on assiste à des tentatives pour les interfacer par une couche de type expertise.

Ces systèmes contiennent une grande quantité d'informations scientifiques exprimées dans une syntaxe de très bas niveau. Ces choix syntaxiques s'expliquent, dans un premier temps, par la vitesse de calcul nécessaire puis par l'inertie du processus de programmation. La communauté du calcul scientifique est bien sûr consciente de ces problèmes, mais la seule tentative des professionnels de l'informatique pour les résoudre a été de proposer de nouveaux langages : Algol, Pascal, APL, ADA, dont l'impact, pour de nombreuses raisons, s'est avéré limité : soit le compilateur n'est pas disponible, soit il est peu performant et le plus souvent ces langages n'apportent pas de fonctionnalités réellement nouvelles, si ce n'est parfois un peu plus de clarté syntaxique. De toute façon, les gains potentiels ne justifient pas la réécriture des programmes existants.

Par contre, la disponibilité sur le marché de langages de calcul formel comme Macsyma rajoute une nouvelle fonctionnalité utile au calcul scientifique, non pas parce que le calcul formel serait un substitut du calcul numérique, il suffit d'avoir pratiqué ces systèmes ou le calcul algébrique pour en voir très rapidement les limites, mais parce qu'il permet d'automatiser une nouvelle couche d'activités en amont du calcul numérique lui-même, faite le plus souvent manuellement : manipulation de formules algébriques ou de programmes vus comme des formules.

Plus en amont encore, le codage de théorèmes et de connaissances empiriques sur le choix de méthodes et d'algorithmes, les aspects interactifs des interfaces homme-machine se codent très bien en Prolog dans la mesure où celui-ci a accès aux facilités fournies par des langages comme Macsyma (entrée et sortie 2D des formules).

En aval, l'édition scientifique et le graphisme 3D complètent la panoplie. Dans ces deux domaines, un langage comme Macsyma apporte des éléments de réponse.

Forts de cette conviction que Macsyma présente un réel intérêt, nous nous efforçons d'automatiser l'ensemble des tâches que doit accomplir l'ingénieur dans une étude du type optimisation de système dynamique depuis la spécification du problème à résoudre jusqu'à la génération automatique de rapports en passant par la génération de programmes numériques et leur exploitation.

La démarche a consisté essentiellement après avoir rassemblé un environnement de programmation adapté à ce type d'activité : Machine Lisp + Macsyma + Oblogis (Prolog) à réaliser les développements suivants :

- la définition et l'écriture d'un petit compilateur permettant la génération de FORTRAN depuis Macsyma. On appelle ce langage MACROFORT,
- la définition et l'écriture d'un autre compilateur permettant la génération de rapport en LATEX à partir de Macsyma appelé MACROTEX,
- le codage en utilisant ces facilités des connaissances du domaine spécifique : l'optimisation de systèmes dynamiques,
- l'écriture d'un éditeur spécialisé à l'entrée des problèmes du domaine,
- l'écriture d'un langage de commande permettant d'interroger le système ou de modifier la base de données associée au problème.

Le système tourne exclusivement pour l'instant sur les machines LISP symbolics. Il est capable de générer des rapports écrits en LATEX résumant une étude complète à partir de la seule spécification du modèle.

Une interface avec un système spécialisé au calcul numérique BASILE est en cours de développement.

Des utilisations pour la résolution de problèmes réels ont été tentées. Elles montrent à la fois l'intérêt d'un tel système et les améliorations à apporter pour qu'une personne non spécialiste du domaine puisse effectivement l'utiliser.

## Références

THEOSYS, *"Commande optimale de Système Stochastique RAIRO"*, Automatique 84.

C. Gomez, J.-P. Quadrat, A. Sulem, *"Towards an Expert System in Stochastic Control: the Hamilton-Jacobi equation Part"*, L.N.C.I.S. n° 63, Springer-Verlag 1984.

C. Gomez, J.-P. Quadrat, A. Sulem, *"Towards an Expert System Control: the Local-feedback Part"*, Congrès Rome sur le contrôle Stochastique, L.N.M. n° 1119, Springer-Verlag 1985.

C. Gomez, J.-P. Quadrat, A. Sulem, *"Computer algebra as a tool for solving optimal control problems. Applications of computer algebra"*, Kluwer Academic Publishers 1985.

J.-P. Chancelier, C. Gomez, J.-P. Quadrat, A. Sulem, *"Un système expert pour l'optimisation de système dynamique"*, Congrès d'Analyse numérique, Versailles, décembre 1985. A paraître North-Holland.

G. Blankenship, C. Gomez, J.-P. Quadrat, A. Sulem, I. Yan, *"An expert system for stochastic control and non-linear filtering 23rd"*, IEEE CDS Las Vegas, décembre 1984.

G. Blankenship, J.-P. Chancelier, C. Gomez, J.- P. Quadrat, A. Sulem, *"An expert system for stochastic control and signal processing"*, MNTS conference, Stockholm, juin 1985.

J.-P. Quadrat, *"Génération automatique de programme numérique en contrôle Stochastique. Calcul Formel pour l'automatique"*, édité par Chenin, 87.

J.-P. Chancelier, C. Gomez, J.-P. Quadrat, *"MACROFORT. A fortran code generator in Macsyma"*, Macsyma letters, 1987.

J.-P. Chancelier, A. Sulem, *"MACROTEX. A latex code generator in Macsyma"*, Macsyma letters, 1988, à paraître.

J.-P. Chancelier, A. Sulem, *"MACROTEX. "Rapport technique"*, INRIA n° 93.

3E PARTIE



# LES ACTIONS INDUSTRIELLES DU GRECO PROGRAMMATION

## RECHERCHE INDUSTRIE *une volonté de coopérer*

La collaboration recherche-entreprise est désormais considérée comme l'un des facteurs les plus importants du développement industriel. Pour l'ensemble des entreprises d'un secteur très concurrentiel comme celui de l'informatique, exigeant un très haut niveau de technicité, marqué de sauts technologiques importants et rapides, la recherche devient aujourd'hui une fonction essentielle.

Depuis quelques années, déjà, de nombreuses structures et procédures, s'appuyant sur des organes publics ou privés, ont vu le jour et s'efforcent de rapprocher le monde de la recherche et celui des entreprises.

Le CNRS s'est défini une politique en faveur de la valorisation et du transfert industriel et mène un certain nombre d'actions. (Mise en place de la direction de la valorisation et des chargés de mission aux relations industrielles, signature de conventions avec de grandes sociétés, création de laboratoires mixtes et de sociétés filiales, etc.)

Dans le cadre de ces nouvelles missions du CNRS et depuis son élargissement en PRC, le GRECO Programmation s'est engagé résolument dans une politique d'ouverture, de dialogue et de coopération avec les milieux professionnels (industriels, SSII, grands utilisateurs) et un certain nombre d'actions qui avaient été lancées dès 1984 vont désormais pouvoir s'intensifier grâce à la mise en place au siège à Bordeaux d'une nouvelle structure chargée de la valorisation et des actions industrielles.

L'élargissement des relations avec le monde industriel implique notamment un nouvel effort d'information et de diffusion de l'information scientifique et technique.

Ce rapport scientifique constitue une mise à jour du précédent document de synthèse *"Bilan et Perspectives"* édité en 1986. Ce dernier, aujourd'hui épuisé, largement diffusé auprès de l'ensemble des responsables de sociétés et de la communauté scientifique universitaire a fortement contribué à faire connaître les activités du GRECO et a permis d'établir de très nombreux contacts.

Ces contacts ont favorisé des actions de rapprochement et permis de créer une synergie nouvelle entre les deux communautés. Des échanges nombreux vont pouvoir s'établir dans un nouvel espace de dialogue convivial et permanent. Les collaborations pourront alors se développer entre partenaires qui auront appris à se connaître, à parler le même langage et à se découvrir des intérêts communs.



Michel Mouyssinat, *Chargé de mission aux actions industrielles GRECO Programmation*

# Une action originale : RIS

## Recherche
## Industrie
## Services

De récentes enquêtes viennent de révéler l'importance de la veille technologique, nouvel enjeu stratégique pour nos industriels. Encouragé par cette récente prise de conscience, le GRECO PROGRAMMATION a mis en place une structure d'accueil d'industriels: RIS (Recherche-Industrie-Services) qui constitue une nouvelle structure d'information et de communication avec le monde industriel.

Ces relations nouvelles avec les entreprises s'inscrivent dans un cadre contractuel. RIS est aussi une convention à laquelle les entreprises peuvent adhérer en acquittant une cotisation annuelle (5.000 francs à 50.000 francs en fonction de leur taille). RIS facilite la nécessaire coopération des entreprises avec le monde de la Recherche en proposant un certain nombre de services.

Veille technologique, formation-recrutement, conseils-assistance, sont les trois volets de l'offre du GRECO PROGRAMMATION. De nombreux services sont gratuits pour les adhérents: participation au cycle annuel des conférences, accès à une liste d'experts, aux logiciels prototypes en ligne, aux réseaux de la recherche et aux publications... D'importantes réductions en outre sont accordées pour la participation à certains stages de formation.

Enfin, à travers le GRECO PROGRAMMATION et l'ensemble de ses équipes (une trentaine environ) réparties dans les principaux laboratoires publics ou privés nationaux, les industriels peuvent établir un contact avec toute la communauté scientifique.

Les premières adhésions sont enregistrées. SLIGOS, CR2A, EDF, SFGL, CODAT INFORMATIQUE sont parmi les principales sociétés impliquées dans ce programme.

Au-delà du désir d'instaurer de nouveaux liens entre les deux communautés et de transférer des connaissances, il y a l'espoir de créer une synergie nouvelle recherche - industrie, qui pourrait déboucher sur des collaborations effectives.

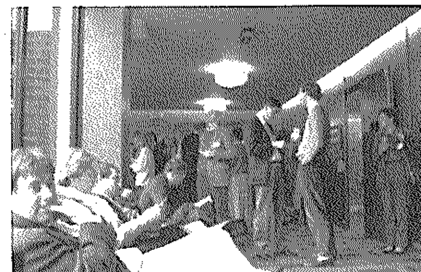
### RIS : VEILLE TECHNOLOGIQUE

Un cycle annuel de conférences (une dizaine environ) organisées à Paris présentent l'état de l'art dans les domaines avancés couverts par les thèmes de recherche du GRECO PROGRAMMATION.

Au cours de l'année 87/88, les techniques et outils en génie logiciel ont fait l'objet d'un effort de promotion tout particulier.

88/89 sera surtout consacrée aux langages et notamment à la programmation par objet.

Réservées aux personnels des sociétés adhérentes à RIS, ces conférences acceptent d'autres participants dans la limite des places disponibles. La participation est gratuite, les demandes d'invitation sont à transmettre au GRECO PROGRAMMATION.



Conférence Génie Logiciel,
*hôtel Méridien-Montparnasse Paris.*

### RIS : ACTIONS DE FORMATION

Les formations initiales dans lesquelles sont impliqués de nombreux chercheurs et ingénieurs du GRECO PROGRAMMATION (à l'Université ou dans les Grandes Ecoles) doivent trouver leurs prolongements nécessaires dans des actions de formation continue.

Un certain nombre de stages de formation sont annuellement mis en place. La participation importante des professionnels aux conférences organisées par le GRECO témoigne de l'intérêt qu'ils portent aux actions de veille technologique et révèle les besoins de formation dans des domaines avancés pour lesquels les concepts, les méthodes et les outils n'appartiennent aujourd'hui encore qu'au milieu de la recherche.

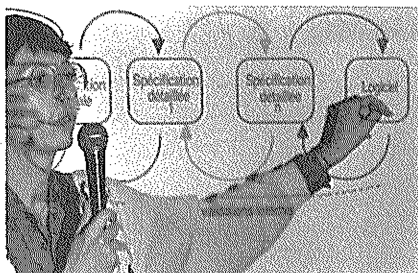
Les nombreux contacts développés avec les milieux professionnels, notamment les sociétés adhérentes à RIS devraient assurer leur succès.

Un soutien financier du FIT (Fonds pour l'Innovation Technologique du Ministère de la Recherche et de l'Enseignement Supérieur) va permettre d'organiser dès 88/89 un ensemble de formations en informatique avancée.

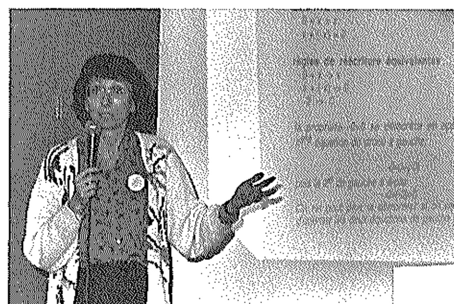
Les thèmes suivants seront abordés: ADA, langage LISP, programmation par objet, spécifications formelles, UNIX, métacompilation.

De nombreuses entreprises parmi lesquelles: AMAIA, SLIGOS, CR2A, EDF, CGE, ORSTOM ont déjà sollicité le GRECO pour élaborer avec lui des programmes de formation, pour elles-mêmes ou leurs clients.

La formation, vecteur du transfert industriel des connaissances, constitue avant tout un moyen privilégié de rapprochement recherche-industrie et devrait susciter de nouvelles collaborations.



**Les Spécifications Formelles :** *leur impact au cours des différentes étapes du cycle de vie du logiciel.*



**Marie-Claude Gaudel,** *directrice du LRI-Orsay, conférence Génie Logiciel.*



**RIS : veille technologique, un auditoire nombreux et attentif.**

## Le Fonds pour l'Innovation Technologique - FIT

*Le développement industriel, la création de nouveaux produits, la conquête de nouveaux marchés passent par la maîtrise des nouvelles technologies et la formation des hommes.*
*Si les stages d'initiation aux technologies modernes se sont multipliés, les formations aux technologies les plus avancées, celles qui s'appuient sur les progrès de la recherche, sont rares et mal connues.*
*Elles sont pourtant essentielles pour que la formation, au-delà de son rôle traditionnel, devienne un vecteur du transfert technologique.*
*Le fonds pour l'innovation technologique -FIT- programme créé conjointement par la direction des enseignements supérieurs (ministère de l'Éducation nationale) et la délégation à la formation professionnelle (ministère des Affaires sociales et de l'Emploi) est maintenant élargi aux ministères de l'Industrie et de l'Agriculture.*
*Ce programme vise à promouvoir des actions de formation continue de haut niveau facilitant le transfert technologique des laboratoires et centres de recherche vers l'industrie.*
*Les actions "FIT" sont destinées aux ingénieurs et techniciens supérieurs des entreprises; elles sont de courte durée, animées par des équipes scientifiques renommées et conduites en collaboration avec les milieux industriels pour répondre à leurs besoins.*
*Ces formations présentent le grand avantage d'être au confluent des formations générales supérieures de base, des derniers résultats de la recherche avancée tant fondamentale qu'appliquée, et également des grandes découvertes scientifiques, techniques et technologiques.*
*Les formations mises en œuvre par le GRECO Programmation répondent à ces objectifs de formation et de veille technologique par une coopération approfondie entre le monde de la recherche et le monde industriel.*
*Avec l'initiative du GRECO Programmation, c'est la première fois qu'un GRECO participe au programme FIT et a pu être soutenu par les pouvoirs publics pour la formation professionnelle. Il reste à espérer que cet exemple puisse être largement suivi et que le GRECO Programmation apparaisse ainsi comme un véritable pionnier.*

*Ministère de la Recherche et de l'Enseignement Supérieur*
*Direction des Enseignements Supérieurs*

# Les projets communautaires

**Esprit**

**Race**

**Comett**

Les programmes de développement technologique lancés par la Communauté Européenne, notamment ESPRIT, le plus important, ont offert de nouveaux modes d'actions favorisant le rapprochement RECHERCHE-INDUSTRIE.

Ces projets, par la dimension européenne des collaborations et la présence d'industriels, ont donné aux équipes de recherche impliquées, une ouverture nouvelle et imposé un champ d'investigation plus large avec une prise en compte des réalités près du terrain industriel. Ils ont fortement influencé le contenu des travaux des chercheurs, leurs modes de travail et offert de nouveaux moyens pour conforter et valider les orientations qu'ils avaient choisies.

Ils ont permis en outre aux équipes de recherche contractantes de trouver de nouvelles sources de financement et ainsi de planifier leurs travaux sur une durée qui devait leur assurer de meilleurs résultats.

Les équipes du GRECO PROGRAMMATION sont engagées nombreuses dans ces actions communautaires, mais les projets présentés ici n'en rendent compte que de façon partielle. En effet, au moment de la préparation de ce document, plusieurs propositions en cours de soumission en réponse à l'appel d'offres ESPRIT 2 devraient donner lieu à de nouveaux contrats.

## PROJETS ESPRIT

**ALPES** *EQUIPE MODAL*
*page 28*

ALPES (Advanced Logic Programming EnvironmentS) est destiné à définir et implémenter un environnement de programmation pour Prolog. Dans l'architecture à trois niveaux de cet environnement (programme, compilation, exécution), l'équipe Modal intervient plutôt au niveau statique du programme. Cela regroupe la définition de primitives de contrôle de types et coercitions, de mise au point, etc.
*partenaires :* CRIL, Bull, Enidata (Bologne Italie), LRI Orsay, Université de Lisbonne, Technische Universitaet Muenchen (Munich RFA).

**CHAMELEON**
*EQUIPE ARCHITECTURE*
*page 81*

Ce projet vise à étudier et réaliser "gracieusement" la migration dynamique de processus dans un réseau de machines et systèmes hétérogènes. Les méthodes utilisées sont aussi bien de nature interprétative que compilatoire, en particulier le projet s'attache à décrire précisément la sémantique des appels systèmes de plusieurs systèmes différents afin de pouvoir les traduire automatiquement d'un système vers un autre.
*partenaires :* Non Standard Logics, Delphi (Italie), Harlequin (Cambridge, Grande-Bretagne).

**COCOS** *EQUIPE ATTRISEM*
*page 26*

COCOS (COmponents for future COmputing Systems) étudie une architecture ouverte pour une station de travail à usage bureautique ainsi que les composants s'y intégrant. L'équipe participe surtout aux tâches autour du langage PARLOG de programmation logique parallèle. Ainsi, PARLOG est implémenté sur une base matérielle associant un processeur symbolique à un processeur classique (68020 ou RISC). Enfin, de nouveaux concepts comme la programmation orientée objet sont développés avec PARLOG.

**FOR-ME-TOO** *EQUIPE LPG*
*page 41*

FOR-ME-TOO (FORmalism - MEthods - TOOls: Software development based on and supporting the concept of reusability of components) est proche de Replay. L'équipe LPG a défini un ensemble de primitives de structuration algébrique ayant servi à la réalisation d'un gestionnaire de fenêtres. Ces pri-

mitives peuvent être considérées comme une extension du projet LPG dans le sens d'une programmation par composants, avec structuration horizontale et verticale.
*partenaires :* Syseca, Division RTT d'Infigenie.

**PROLOG** *EQUIPE PROLOG-III*
*page 66*

Le GIA est engagé dans le projet P1219 "Further development of Prolog and its validation by KBS in technical areas". Il s'agit concrètement de réaliser un système expert en diagnostic de panne de voitures grâce à une version améliorée de Prolog II. Le GIA devra en particulier ajouter à la version existante une arithmétique complète et l'intégration des contraintes numériques à l'unification. Par ailleurs, les deux industriels allemands synthétiseront une base de connaissances et les règles d'expertise à partir de leur savoir-faire dans le domaine. Ces points serviront de base au futur système.
*partenaires :* PrologIA, Daimler-Bentz (Stuttgart RFA), Bosch (RFA).



**RAGTIME** *EQUIPE MANENS*
*page 60*

RagTime (Reusable components and Automated Generation of real-TIME implementations) cherche à construire un logiciel de haut niveau pour la spécification de systèmes intégrant temps-réel et parallélisme. Le groupe Manens participe à la préparation de ce projet ESPRIT-2 et devrait intervenir autour du langage S3L : Comparaison à SETL, évaluation de l'implémentation, intégration de possibilités de concurrence vers la programmation relationnelle.
*partenaires :* Thomson-CSF.

**REPLAY** *EQUIPE SPRAC & LPG*
*page 46 & page 41*

Replay est le successeur de Tool-Use cité plus bas. Son rôle est d'estimer l'impact des outils de manipulation du développement dans un contexte de réutilisation du logiciel. Deux pôles sont privilégiés : la composition de plans de développement avec entre autres les aspects réutilisation descendante et assemblage ascendant.
Le contrôle permanent des propriétés assurant la vérification des contraintes opérationnelles du cahier des charges.
*partenaires :* CISI-Engenierie, UCL (Louvain Belgique), Alpha-SAI (Athènes Grèce), CRI A/S (Copenhague Danemark), E2S (Belgique).

**SOMIW** *EQUIPE LANGAGES-OBJETS*
*page 52*

SOMIW (Secure Open Multimedia Integrated Workstation) sera une station de travail bureautique d'une puissance entre un PC-AT et un Sun. Le groupe Langages-Objets intervient dans le projet Somiw par le biais de Philippe Gaudron qui réalise le système SOS à objets répartis. Ce travail, fortement basé sur C++, a engendré un éditeur de liens dynamique permettant la migration d'objets. SOS est ainsi envisagé comme plate-forme d'expérimentation d'un univers multimédia distribué reposant sur l'approche objet.
*partenaires :* Bull

**TIPE** *EQUIPE EURECA*
*page 20*

TIPE représente les concepts Types, Inheritance, Polymorphism & Equalities. Ce projet qui sera soumis au programme ESPRIT-2 est encore en cours de définition. Il visera à étudier et réaliser un environnement de programmation sûr, destiné au développement d'applications d'aéronautique ou à haut risque. L'environnement de preuve intégrera en particulier un langage de spécification puissant et exécutable.

**TOOL-USE** *EQUIPE SPRAC*
*page 46*

Ce projet vise à fournir une assitance active à la conception, à l'implémentation et l'évolution du logiciel. Cela passe par quatre points fondamentaux : étude et compréhension des méthodes de développement, formalisation du processus de développement, définition et réalisation d'un langage de développement (DEVA), réalisation de l'environnement support. Actuellement, DEVA permet de spécifier des développements simples et un prototype est en cours d'achèvement.
*partenaires :* CISI-Engenierie, Trinity College (Dublin Irlande), Generics (Irlande), UCL (Louvain Belgique), GMD (Karlsruhe RFA), Biomatic (Freiburg RFA)

**04** *ÉQUIPE RANK XEROX FRANCE, P. Cointe, M. Politis ; Contractant principal : Bull.*

Le projet 04 aura pour point de départ l'utilisation des postes de travail et l'état de l'art de la recherche dans le domaine en 88. Les travaux seront menés de 1989 à 1993 en intégrant les développements issus de la pratique et de la recherche. Les résultats conduiront à un poste de travail bureautique multi-média à l'usage des non-informaticiens.
Pour ce faire, il sera fait un grand usage de la programmation par objets (voir page 79) et plus particulièrement des systèmes CommonLisp-CLOS et BETA.

## PROJET RACE

**SPECS** *ÉQUIPE OASIS DU CNET*
*page 44*

L'équipe participe dans le cadre de la phase principale du programme européen RACE, au projet SPECS concernant les environnements de spécification pour les systèmes de communication. Ces travaux ont pour échéance fin 1992.

## PROJET COMETT

*EQUIPE MAIDAY*
*page 38*

Une mise en œuvre des outils développés par l'équipe du projet MAIDAY est à l'étude au plan industriel dans le cadre de la participation du projet européen COMETT.

# INDEX & ÉQUIPES ET LABORATOIRES DE RATTACHEMENT

# Index

*PAGES* **A** *MOTS-CLÉS*

# Equipes & laboratoires de rattachement

*PAGES* **B** *LABORATOIRES*

18, 80 ***Université de Bordeaux I***
U.E.R. de Mathématiques et Informatique
351, cours de la Libération
33405 Talence Cedex
☎ 56 84 60 89 & 56 84 60 90
P. Castéran, R. Cori, B. Courcelle.

52 ***Université de Brest***
Université de Bretagne Occidentale
6, avenue V.-Le-Gorgeu
☎ 98 03 16 94
J. Bézivin.

18 ***Bull***
68, route de Versailles
78430 Louveciennes
☎ 39 02 42 11
J.-M. Kerisit, J.-M. Pugin

**C**

28 ***Université de Caen***
C.E.R. de Mathématiques
Esplanade de la Paix
14032 Caen Cedex
☎ 31 94 81 40
P. Enjalbert.

56 ***C.G.E.***
Laboratoires de Marcoussis
Division Informatique
Route de Nozay
91460 Marcoussis
☎ (1) 64 49 11 51
C. Benoit.

44 ***C.N.E.T.***
Dép. "Logiciel et système de commutation"
38, rue du Général-Leclerc
92131 Issy-Les-Moulineaux Cedex
☎ (1) 45 29 44 44 - 45 29 47 65
G. Barberye.

**E**

30 ***École Normale Supérieure***
Département Mathématiques & Informatique
45, rue d'ULM
75005 Paris
☎ 43 29 12 25
Guy Cousineau, C. Puech.

86 ***École Polytechnique***
Centre de Mathématiques Appliquées
Plateau de Palaiseau
91128 Palaiseau Cedex
☎ (1) 60 19 40 91
J.-M. Steyaert.

**G**

41 ***Université de Grenoble***
I.M.A.G.-L.I.F.I.A./Université de Grenoble
46, avenue Félix-Viallet
38031 Grenoble Cedex
☎ 76 57 45 00 & 76 57 46 64
D. Bert.

**I**

66, 79 ***I.B.M.***
Centre Scientifique
36, avenue Raymond-Poincaré
75016 Paris
☎ (1) 45 05 14 00 & 42 96 14 75
M. Gillet, P. Bellot.

18, 26, 30, 87 ***I.N.R.I.A. (Rocquencourt)***
Domaine de Voluceau
Rocquencourt
78150 Le Chesnay
☎ (1) 39 63 55 11
G. Huet, P. Deschamps, J.-P. Quadrat, P. Deransart, P. Flajolet, B. Lang, Y. Bekkers.

26 ***I.N.R.I.A. (Sophia-Antipolis)***
Sophia-Antipolis
06560 Valbonne
☎ 93 95 44 24 ISI 93 95 44 44 & 93 65 77 77
G. Kahn, P. Franchi-Zannettachi.

**L**

18 ***Université de Lille***
UER IEAA - M3
59655 Villeneuve-d'Ascq
☎ 20 43 47 22
J.-P. Delahaye.

**M**

66 ***Université de Marseille***
Groupe d'Intelligence Artificielle
Université de Marseille II Luminy
70, route Léon-Lachamp
13288 Marseille Cedex 2
☎ 91 26 90 00
A. Colmerauer, F. Giannesini.

84 ***Université de Montpellier***
C.R.I.M.
Université du Languedoc
860, route de Saint-Priest
34100 Montpellier
☎ 67 63 04 60
J.-M. Boé.

## N

20, 38 **_Université de Nancy_**
C.R.I.N./Université de Nancy
B.P. 239
54506 Vandœuvre-Lès-Nancy Cedex
☎ 83 91 21 19
P. Lescanne, J.-L. Rémy, J.-P. Finance,
J. Guyard.

## O

46 **_ONERA-CERT_**
2, rue Édouard-Belin
31055 Toulouse Cedex
☎ 61 55 71 11
R. Jacquart, M. Lemoîne.

18, 26, 60 **_Université d'Orléans_**
U.E.R. de Mathématiques et Informatique
Rue de Chartres
45046 Orléans Cedex
☎ 38 63 22 16
Lorho, C. Gresse, Lacrampe.

34, 81 **_Université d'Orsay_**
L.R.I. Bât. 490
Université de Paris Sud
91405 Orsay
☎ (1) 69 41 66 29 & 69 41 66 28
M. Bidoit, J.-P. Jouannaud, I. Filotti,
C. Puech, C. Gresse.

## P

62,63 **_Université de Paris VI_**
L.I.T.P.
4, place Jussieu
75230 Paris Cedex 05
☎ (1) 43 25 98 74
E. Saint-James, J.-F. Perrot,
P. Cointe, C. Queinnec.

52 **_Université de Paris VI_**
LAFORIA
4, place Jussieu, tour 45-55
75005 Paris
J.-F. Perrot.

72, 84 **_Université de Paris VII_**
L.I.T.P.
2, place Jussieu
75221 Paris Cedex
☎ (1) 43 36 25 25
D. Perrin.

72 **_Université de Paris VIII_**
U.E.R. d'Informatique
2, rue de la Liberté
93526 Saint-Denis Cedex 02
☎ (1) 48 21 63 64
P. Greussay, H. Wertz.

## R

52 **_Rank Xérox France_**
DRDBI
12, place de l'Iris
92071 Paris la Défense Cedex 38
☎ 47 62 11 39
P. Cointe.

76 **_Université de Rennes_**
I.R.I.S.A.
Campus de Beaulieu
35042 Rennes Cedex
☎ 99 36 20 00
Y. Bekkers.

84 **_Université de Rouen_**
Laboratoire d'Informatique
Place Émile-Blondel
76130 Mont-Saint-Aignan
☎ 35 98 28 50
J.-P. Pécuchet, C. Choffrut.

## S

26 **_Université de Strasbourg_**
Département d'Informatique
Université de Strasbourg
7, rue René-Descarts
67084 Strasbourg Cedex
☎ 88 61 48 20
J. Françon, J.-M. Schramm.

## T

28, 50, 64 **_Université de Toulouse, Paul-Sabatier_**
L.S.I. - ENSEEIHT
118, route de Narbonne
31062 Toulouse Cedex
☎ 61 55 69 65 - 61 58 83 83
J. Vignolle, P. Salle, C. Percebois,
Fariñas del Cerro. Beaufils.

**Fiches de Liaison**

**A retourner au**

GRECO Programmation du CNRS
351, cours de la Libération
33405 Talence Cedex
☎ 56 84 60 89 / 56 84 60 90
Télécopie : 56 80 08 37

A partir de janvier 89, vous pourrez interroger notre service Télétel et communiquer avec le GRECO Programmation.

Tapez : 3616 code GRECOTEL

1 **SERVICES**

Bon de commande
Nouveaux enjeux pour la recherche & ses applications

2
Convention RIS

3 **CALENDRIER**

Veille Technologique
Invitation aux conférences

4 **FORMATION**

Stages de formation
Langage PROLOG
Langage LISP
Langage objets
Langage ADA
Génie logiciel (spécifications formelles)
Système UNIX (introduction)
UNIX avancé
Langage C
Autres demandes :

## BON DE COMMANDE

1

NOUVEAUX ENJEUX POUR LA RECHERCHE & SES APPLICATIONS

NOM ______________________

SOCIÉTÉ ______________________

ADRESSE ______________________

Je vous prie de bien vouloir m'envoyer à l'adresse ci-dessus ________ exemplaire(s) du document à 230 francs ttc l'unité, frais d'envoi inclus

Ci-joint règlement sous la forme de ☐ chèque ☐ CCP.

d'un montant de __________

## CONVENTION RIS

2

NOM ______________________

PRÉNOMS ______________________

SOCIÉTÉ ______________________

ADRESSE ______________________

__________ ☎ __________

souhaite recevoir à titre d'information un exemplaire de la convention RIS

## VEILLE TECHNOLOGIQUE

3

INVITATION AUX CONFÉRENCES

NOM ______________________

PRÉNOMS ______________________

SOCIÉTÉ ______________________

ADRESSE ______________________

__________ ☎ __________

souhaite participer au cycle de conférences du GRECO Programmation du CNRS et recevoir le programme

## STAGES DE FORMATION

4

NOM __________ PRÉNOMS __________

SOCIÉTÉ ______________________

ADRESSE ______________________

__________ ☎ __________

est intéressé(e) par les stages suivants :

- ☐ Langage PROLOG
- ☐ Langage LISP
- ☐ Langage objets
- ☐ Langage ADA
- ☐ Génie logiciel (spécifications formelles)
- ☐ Système UNIX (introduction)
- ☐ UNIX avancé
- ☐ Langage C
- ☐ Autres demandes __________

Recherche Industrie

GRECO PROGRAMMATION DU CNRS

Bordeaux

Recherche Industrie

GRECO PROGRAMMATION DU CNRS
Université de Bordeaux I
351, cours de la Libération
33405 Talence Cedex

Recherche Industrie

GRECO PROGRAMMATION DU CNRS
Université de Bordeaux I
351, cours de la Libération
33405 Talence Cedex

Recherche Industrie

GRECO PROGRAMMATION DU CNRS
Université de Bordeaux I
351, cours de la Libération
33405 Talence Cedex

Centre National de la Recherche Scientifique





351, cours de la Libération 33405 Talence Cedex
[redacted] - Télécopie 56 80 08 37
Adresse réseau : greco @ [redacted]
☎ 56.84.60.89 - 60.90 - gréco-prog. sr

PRIX 230 F TTC